Modeling, Annotating, and Querying Geo-Semantic Data Warehouses

Gür, Nurefsan

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

MODELING, ANNOTATING, AND QUERYING GEO-SEMANTIC DATA WAREHOUSES

BY
NUREFŞAN GÜR

DISSERTATION SUBMITTED 2020

AALBORG UNIVERSITY
DENMARK

Modeling, Annotating, and Querying Geo-Semantic Data Warehouses

Ph.D. Dissertation
Nurefşan Gür

Dissertation submitted January, 2020

A thesis submitted to the Technical Faculty of IT and Design at Aalborg University (AAU) and the Faculty of Engineering at Université Libre de Bruxelles (ULB), in partial fulfilment of the requirements within the scope of the IT4BI-DC programme for the joint Ph.D. degree in Computer Science. The thesis is not submitted to any other organization at the same time.

Dissertation submitted: January, 2020

AAU PhD Supervisor: Prof. Torben Bach Pedersen
Aalborg University, Denmark

AAU PhD Co-Supervisor: Prof. Katja Hose
Aalborg University, Denmark

ULB PhD Supervisor: Prof. Esteban Zimányi
Université Libre de Bruxelles, Belgium

PhD committee: Associate Professor Kristian Torp (chairman)
Aalborg University

Professor Lars Harrie
GIS Centre, Lund University

Researcher Sandro Bimonte
National Research Institute of Science and Technology for Environment and Agriculture (IRSTEA)

ULB PhD Committee: Assoc. Prof. Stijn Vansummeren
Université Libre de Bruxelles, Belgium

Assoc. Prof. Mahmoud Sakr
Université Libre de Bruxelles, Belgium

Assoc. Prof. Kristian Torp
Aalborg University, Denmark

Professor Lars Harrie
GIS Centre, Lund University

Dr. Sandro Bimonte, National Research Institute of Science and Technology for Environment and Agriculture - IRSTEA, France

PhD Series: Technical Faculty of IT and Design, Aalborg University
Department: Department of Computer Science
ISSN (online): 2446-1628
ISBN (online): 978-87-7210-587-1

Published by:
Aalborg University Press
Langagervej 2 | DK – 9220 Aalborg Ø
Phone: +45 99407140 | aauf@forlag.aau.dk | forlag.aau.dk

Printed in Denmark by Rosendahls, 2020

Abstract

Due to recent advances in Semantic Web (SW) technologies and world-wide movement on publishing Linked Open Data (LOD) by following a set of principles on the SW, many governmental organizations and public agencies have been publishing large volumes of geospatial data on the SW. This large amount of spatial data on the SW yields a need for advanced analysis. In the traditional database world, data warehouses and On-Line Analytical Processing (OLAP) is proven to be a well constructed and efficient way of storing and querying massive amounts of data with advanced analytical perspectives. Current state-of-the-art SW technologies support only multi-dimensional annotation of data warehouses and querying with traditional OLAP operators over non-spatial data. To reveal new perspectives over SW data, it is important to utilize the spatial information of the data sets being published, while modeling and querying the (SW) data warehouses. Even though there are many studies around the geospatial SW, none of them addresses the multi-dimensional aspects of spatial data being published or support querying with spatial OLAP (SOLAP) operators. We believe a large number of spatial data sets on the SW can be utilized with their full potential by addressing the lack of methods, tools, and applications of spatial data warehouses on the SW. Therefore the thesis addresses several challenges related to geo-semantic data warehouses to model, query, and derive analytical perspectives over a huge amount of geo-semantic data with SOLAP operators and spatial multi-dimensional enrichment of SW data.

This thesis first presents the best practices for publishing Danish governmental data sets from agricultural, spatial, and business domains on the SW as an initial effort to publish open governmental data as Linked Open Data in Denmark. The published LOD endpoint is queried with standard and aggregate query templates and evaluated against three different query processing scenarios with native RDF, relational and virtual strategies. Query performance for aggregated query templates on native RDF demonstrated significantly faster response time than the other strategies. The results establish a promising basis for geo-semantic data warehouses, as aggregate queries are fundamental in multi-dimensional models and in data warehousing.

Next, the thesis proposes a multi-dimensional cube vocabulary for SOLAP - *QB4SOLAP* as an initial step of a foundation for geo-semantic data warehouses. The thesis defines full formalizations of multi-dimensional spatial concepts from QB4SOLAP (such as spatial dimension hierarchies, spatial measures, aggregate functions, topological relations) in RDF format. By using the QB4SOLAP vocabulary, spatial data can be annotated and published on the SW with complete spatial multi-dimensional concepts, which were not available before. Spatial multi-dimensional annotation of RDF data with QB4SOLAP overcomes several limitations of spatial analytical querying of SW data with a set of SOLAP operators (e.g., s-roll-up and s-slice), where SOLAP operators reveal new analytical perspectives to the users. SOLAP operators create a dynamic interpretation of the multi-dimensional concepts (i.e., dynamic spatial hierarchy) by employing spatial functions (e.g., distance) or topological relations (e.g., within). Due to the fact that data warehouse queries typically involve nesting of SOLAP operators, which makes it almost impossible for non-SW-experts to formulate nested SOLAP queries in native RDF query language (SPARQL), the thesis proposes algorithms for translating individual and nested SOLAP queries from high-level multi-dimensional expressions into SPARQL query syntax.

Furthermore, using the generation algorithms and formalizations of QB4SOLAP and SOLAP operators, a geo-semantic OLAP tool for the SW - *GeoSemOLAP* is introduced to remove the entry barrier for data warehouse users to query geo-semantic data warehouses without knowledge of RDF/SPARQL. The GeoSemOLAP tool is tested and demonstrated with a non-trivial spatial multi-dimensional use case, which is modeled with the QB4SOLAP vocabulary. By using the QB4SOLAP multi-dimensional schema definitions, GeoSemOLAP parses the use case data with corresponding spatial multi-dimensional concepts, such as spatial levels, spatial attributes, and spatial measures, which can be used as input parameters to the SOLAP operations from an intuitive and easy-to-use graphical user interface that is provided with interactive maps.

The QB4SOLAP vocabulary is validated on several use cases including Danish governmental, environmental, agricultural, farming, and spatial data sets, which can be annotated with spatial multi-dimensional concepts of QB4SOLAP. By applying QB4SOLAP on complex real-world data sets from various organizations and resources, a comprehensive insight is brought on to spatial and multi-dimensional modeling of governmental data with best practices, discussions, and perspectives. In order to exploit the data with new analytical perspectives that were not available before, the use cases are queried with a set of common (individual and nested) SOLAP operators in SPARQL.

Finally, the thesis addresses the lacking automated methods for spatial multi-dimensional annotations on the Semantic Web and proposes an enrich-

ment framework namely RDF2SOLAP to enrich the existing RDF data cubes. The thesis proposes a set of hierarchical enrichment algorithms and a set of factual enrichment algorithms within the scope of the RDF2SOLAP enrichment process. In each set, there are algorithms for detecting explicit relations and discovering implicit relations between spatial level members (hierarchical enrichment) and between fact-level members (factual enrichment). Moreover, in factual enrichment, the fact schema is automatically redefined from the outcome of the instance level enrichment algorithms over fact members and level members. RDF2SOLAP enrichment process allows spatial data warehouse users to query the existing spatial RDF endpoints with SOLAP operators without needing to download, convert, model and store the data in an offline spatial data warehouse. Experimental evaluation results show that the proposed framework demonstrates the efficient processing of the enrichment algorithms directly on the RDF data.

Resumé

På grund af de seneste fremskridt omkring Semantic Web (SW) teknologier og den verdensomspændende bevægelse for at udgive Linked Open Data (LOD) ved at følge et sæt af principper på det semantiske web (SW), har mange regeringsorganisationer og offentlige myndigheder udgivet store mængder geo-spatielle data på det semantiske web. Denne store mængde af spatielle data kræver avanceret analyse. I den traditionelle database-verden er datavarehuse og On-Line Analytical Processing (OLAP) anerkendt for at være en velstruktureret og effektiv metode for at gemme og forespørge på enorme mængder af data med et avanceret analytisk perspektiv. De seneste SW teknologier understøtter kun multi-dimensionel annotering af datavarehuse og forespørgsler med traditionelle OLAP operatorer over ikke-spatiel data. For at kunne understøtte nye perspektiver og avanceret (spatiel) analyse over SW data, når man modellerer og forespørger på (SW) datavarehuse, er det vigtigt at anvende de datasæt med spatiel information, der bliver udgivet imens. Selvom der er mange studier vedrørende det geo-spatielle semantiske web, er der ingen der adresserer det multi-dimensionelle aspekt af spatiel data, der bliver udgivet eller understøtter forespørgsel af spatiel OLAP (SOLAP) operatorer. Vi mener at mange spatielle data sæt på det semantiske web man blive anvendt til deres fulde potentiale ved at adressere manglen på metoder, værktøjer og applikationer af datavarehuse på det semantiske web. Derfor adresserer afhandlingen adskillige udfordringer relateret til geo-semantiske datavarehuse til at modellere, forespørge på og aflede analytiske perspektiver over enorme mængder af geo-semantiske data.

Først præsenterer afhandlingen den bedste praksis for at udgive danske offentlige datasæt fra landbrugs-, spatielle- og forretningsdomæner på det semantiske web som en initiel indsats til at udgive offentlige regeringsdata som Linked Open Data (LOD) i Danmark. De udgivne LOD endpoint bliver forespurgt på standard og aggregerede forespørgselsskabeloner og evalueret mod tre forskellige forespørgselsprocesseringsscenarier med RDF, relationelle og visuelle strategier. Effektiviteten af forespørgslerne for aggregerede forespørgselsskabeloner på RDF demonstrerede betydelig hurtigere responstid end andre strategier. Resultaterne etablerer en lovende basis for geo-semantis-

ke datavarehuse som aggregerede forespørgsler, da aggregerede forespørgsler er fundamentale i multi-dimensionelle modeller og i datavarehusning.

Derefter foreslår afhandlingen en multi-dimensionel kubeontologi for SOLAP, QB4SOLAP, som det første skridt for et fundament for geo-semantiske datavarehuse. Afhandlingen definerer fuld formalisering af multi-dimensionelle spatielle koncepter fra QB4SOLAP (fx. spatielle dimensionshierarkier, spatielle målinger, aggregerede funktioner og topologiske relationer) i RDF format. Ved at anvende QB4SOLAP ontologien, kan spatielle data blive annoteret og udgivet på det semantiske web med komplette spatielle multidimensionelle koncepter, som ikke tidligere var tilgængelige. Spatielle multidimensionelle annoteringer af RDF dataen med QB4SOLAP overvinder adskillige begrænsninger med spatielle analyseforespørgsler af SW data med et sæt af SOLAP operatorer (fx. s-roll-up og s-slice), hvor SOLAP operatorerne giver nye analytiske perspektiver for brugerne. SOLAP operatorerne laver en dynamisk fortolkning af de multi-dimensionelle koncepter (dvs. et dynamisk spatielt hierarki) ved at anvende spatielle funktioner (fx. afstand) eller topologiske relationer (fx. indenfor). Pga. at datavarehusforespørgsler typisk involverer indlejring af SOLAP operatorer, som gør det næsten umuligt for ikke-SW-eksperter at formulere indlejrede SOLAP forespørgsler på RDF forespørgselssprog (SPARQL), foreslår afhandlingen algoritmer til at oversætte individuelle og indlejrede SOLAP forespørgsler fra højniveau multi-dimensionelle udtryk til SPARQL forespørgselssyntaks.

Yderligere, ved at anvende generering af algoritmer og formaliseringer af QB4SOLAP og SOLAP operatorer, introduceres et geo-semantisk OLAP værktøj for SW, GeoSemOLAP, som fjerner de indgangsbarrierer datavarehus brugere har i forhold til at forespørge geo-semantiske datavarehuse uden viden om RDF/SPARQL. GeoSemOLAP værktøjet er testet og demonstreret med et ikke-trivielt spatielt multi-dimensionelt eksempel for anvendelse, som er modelleret med QB4SOLAP ontologien. Ved at anvende QB4SOLAPs multi-dimensionelle skemadefinitioner, fortolker GeoSemOLAP dataen for anvendelseseksemplet med tilsvarende spatielle multi-dimensionelle koncepter, som spatielle niveauer, spatielle attributter og spatielle målinger, som kan blive anvendt som inputparametre til SOLAP operationerne fra en intuitiv og letanvendelig grafisk brugerflade som er forsynet med interaktive kort.

QB4SOLAP ontologien er valideret på adskillige eksempler for anvendelse. Disse inkluderer offentlige, miljømæssige, landbrugs og spatielle data sæt, som kan blive annoteret med spatielle multi-dimensionelle koncepter for QB4SOLAP. Ved at anvende QB4SOLAP på komplekse datasæt fra den virkelige verden fra forskellige organisationer og ressourcer, er en omfattende indsigt opnået på spatiel og multi-dimensionel modellering af offentlig data med bedste praksis, diskussioner og perspektiver. For at udnytte dataen med nye analytiske perspektiver, der ikke tidligere var tilgængelige, er anvendelseseksemplerne forespurgt på et sæt af fælles (individuelle og indlejrede) SOLAP

operatorer i SPARQL.

Til slut adresserer afhandlingen manglen på automatiserede metoder for spatiel multi-dimensionel annotering på det semantiske web og foreslår et berigelsessystem, RDF2SOLAP, til at berige de eksisterende RDF datakuber. Afhandlingen foreslår et sæt af hierarkiske berigelsesalgoritmer og et sæt af faktuelle berigelsesalgoritmer inden for rammerne af RDF2SOLAP berigelsesprocessen. I hvert sæt er der algoritmer til at detektere eksplicitte relationer og til at detektere implicitte relationer mellem spatielle niveaumedlemmer (hierarkisk berigelse) og mellem fakta-niveaumedlemmer (faktuel berigelse). Derudover er faktaskemaet i faktuel berigelse automatisk redefineret ud fra udfaldet af instansniveau berigelsesalgoritmer over faktamedlemmer og niveaumedlemmer. RDF2SOLAP berigelsesprocessen tillader spatielle datavarehus brugere at forespørge på eksisterende spatielle RDF endpoints med SOLAP operatorer uden at være nødsaget til at hente, konvertere, modellere og gemme dataen i et offline spatielt datavarehuse. Eksperimentelle evalueringsresultater viser, at det foreslåede system demonstrerer effektiv processering af berigelsesalgoritmerne direkte på RDF dataen.

Résumé

En raison des récents progrès des technologies du Web sémantique (WS) et du mouvement mondial de publication de données ouvertes liées (Linked Open Data ou LOD en anglais) en suivant un ensemble de principes sur le WS, de nombreuses organisations gouvernementales et agences publiques ont publié de grands volumes de données géospatiales sur le WS. Un telle quantité de données géospatiales sur le WS entraîne un besoin d'analyse avancée. Dans le domaine des bases de données, les entrepôts de données (Data Warehouses en anglais) et le traitement analytique en ligne (OnLine Analytical Processing ou OLAP en anglais) se sont avérés être un moyen bien conçu et efficace de stocker et d'interroger des quantités massives de données avec des perspectives avancées d'analyse. Les technologies logicielles de pointe actuelles ne prennent en charge l'annotation des entrepôts de données multidimensionnels et l'interrogation avec des opérateurs OLAP traditionnels que sur des données non spatiales. Pour révéler de nouvelles perspectives sur les données du WS, il est important d'utiliser l'information spatiale des jeux de données publiés, tout en modélisant et en interrogeant les entrepôts de données sur le WS. Même s'il existe de nombreuses études sur le WS géospatial, aucune d'entre elles n'aborde les aspects multidimensionels de gestion des données spatiales publiées ni permet d'interroger les données à l'aide d'opérateurs OLAP spatiaux (Spatial OLAP ou SOLAP en anglais). Nous croyons qu'un grand nombre de jeux de données spatiales sur le WS peuvent être utilisés à leur plein potentiel en remédiant au manque de méthodes, d'outils et d'applications des entrepôts de données spatiales sur le WS. Par conséquent, cette thèse aborde plusieurs défis liés aux entrepôts de données géo-sémantiques afin de modéliser, d'interroger et de dériver des perspectives analytiques sur une énorme quantité de données géo-sémantiques avec les opérateurs SOLAP et l'enrichissement spatiale et multidimensionnelle des données sémantiques.

Dans un premier temps, la thèse présente les meilleures pratiques pour la publication de jeux de données gouvernementales danoises sur les domaines agricole, spatial et commercial sur le WS, comme un premier effort pour publier des données gouvernementales ouvertes sous forme de données ouvertes

liées au Danemark. Le point d'accès (endpoint en anglais) LOD est interrogé via des modèles de requête standard et agrégés, et évalué par rapport à trois différents scénarios de traitement de requêtes, à savoir, des solutions RDF natives, des solutions relationnelles, et des solutions virtuelles. Les modèles de requête agrégés sur RDF natif ont démontré un temps de réponse significativement plus rapide que les autres stratégies. Les résultats établissent une base prometteuse pour les entrepôts de données géo-sémantiques car les requêtes agrégées sont fondamentales pour les modèles multidimensionels et pour l'entreposage de données.

Ensuite, la thèse propose un vocabulaire pour des cubes multidimensionnels et SOLAP - *QB4SOLAP* comme étape initiale d'une fondation pour les entrepôts de données géo-sémantiques. La thèse définit des formalisations complètes des concepts spatiaux multidimensionels de QB4SOLAP (tels que les hiérarchies de dimensions spatiales, les mesures spatiales, les fonctions agrégées, les relations topologiques, etc.) en format RDF. En utilisant le vocabulaire de QB4SOLAP, les données spatiales peuvent être annotées et publiées sur le WS avec des concepts multidimensionels spatiaux complets, qui n'étaient pas disponibles auparavant. L'annotation spatiale des données RDF avec QB4SOLAP permet de surmonter plusieurs limitations de l'interrogation analytique spatiale des données sémantiques via un ensemble d'opérateurs SOLAP (p. ex. s-roll-up, s-slice, etc.) qui révèlent de nouvelles perspectives analytiques aux utilisateurs. Les opérateurs SOLAP créent une interprétation dynamique des concepts multidimensionels (c.-à-d. une hiérarchie spatiale dynamique) en employant des fonctions spatiales (p. ex., la distance) ou des relations topologiques (p. ex., à l'intérieur). Du fait que dans les entrepôts de données les requêtes impliquent typiquement l'imbrication des opérateurs SOLAP et que cela rend presque impossible pour les non-experts en entrepôts de données de formuler des requêtes SOLAP imbriquées en langage de requêtes RDF natif (SPARQL), la thèse propose des algorithmes pour traduire des requêtes SOLAP individuelles et imbriquées à partir d'expressions multidimensionnelles de haut niveau en syntaxe SPARQL.

De plus, en utilisant les algorithmes de génération et les formalisations des opérateurs QB4SOLAP et SOLAP, un outil OLAP géo-sémantique pour le WS - *GeoSemOLAP* est proposé pour supprimer la barrière d'entrée pour les utilisateurs d'entrepôts de données sans connaissance de RDF/SPARQL lorsqu'ils interrogent des entrepôts de données géo-sémantiques. L'outil GeoSemOLAP est testé et illustré avec un cas non-trivial d'utilisation de multidimensionels spatial, qui est modélisé avec QB4SOLAP. En utilisant les définitions du schéma multidimensionel de QB4SOLAP, GeoSemOLAP analyse les données du cas d'utilisation avec les concepts multidimensionels spatiaux correspondants, tels que les niveaux spatiaux, les attributs spatiaux et les mesures spatiales, qui peuvent être utilisés comme paramètres d'entrée pour les opérations de SOLAP à partir d'une interface utilisateur graphique

intuitive et facile à utiliser qui est fournie avec des cartes interactives.

Le vocabulaire QB4SOLAP est validé dans plusieurs cas d'utilisation, y compris des jeux de données gouvernementales, environnementales, agricoles et spatiales du Danemark, qui peuvent être annotés avec les concepts multidimensionels spatiaux de QB4SOLAP. En appliquant QB4SOLAP sur des jeux de données complexes du monde réel provenant de diverses organisations et ressources, un aperçu complet est apporté à la modélisation spatiale et de gestion des données gouvernementales avec des meilleures pratiques, discussions et perspectives. Afin d'exploiter les données avec de nouvelles perspectives analytiques, les cas d'utilisation sont interrogés avec un ensemble d'opérateurs SOLAP (individuels et imbriqués) communs en SPARQL.

Enfin, la thèse aborde la manque de méthodes automatisées pour les annotations multidimensionnelles spatiales sur le WS et propose un cadre d'enrichissement, à savoir RDF2SOLAP, pour enrichir les cubes de données RDF existants. La thèse propose un ensemble d'algorithmes d'enrichissement hiérarchique et un ensemble d'algorithmes d'enrichissement factuel dans le cadre du processus d'enrichissement RDF2SOLAP. Dans chaque ensemble, il y a des algorithmes pour détecter les relations explicites et découvrir les relations implicites entre les membres de niveau spatial (enrichissement hiérarchique) et entre les membres de niveau factuel (enrichissement factuel). De plus, dans l'enrichissement factuel, le schéma de faits est automatiquement redéfini à partir du résultat des algorithmes d'enrichissement au niveau de l'instance sur les membres d'un fait et les membres d'un niveau. Le processus d'enrichissement RDF2SOLAP permet aux utilisateurs de l'entrepôt de données spatiales d'interroger les « endpoints » RDF spatiaux existants avec les opérateurs SOLAP sans avoir besoin de télécharger, convertir, modéliser et stocker les données dans un entrepôt de données spatiales hors ligne. Les résultats de l'évaluation expérimentale montrent que le cadre proposé démontre l'efficacité du traitement des algorithmes d'enrichissement directement sur les données RDF.

Acknowledgements

I would like to thank to a number of people who have helped and supported me during the years of pursuing my Ph.D. studies.

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Torben Bach Pedersen for giving me the opportunity to work with him. I would like to thank him for his constructive feedback, continuous support, and motivation. He has always been patient and available to help me under any circumstance, which I am very grateful for. Besides his professional and technical guidance, his motivational advice helped me during the research process and completing the writing of this thesis. Next, I would like to thank my co-supervisor Prof. Katja Hose for her invaluable support during my Ph.D. studies. She has been a great advisor and a mentor for improving the quality and presentation of the whole research. I would also like to thank Prof. Esteban Zimányi as my co-supervisor at Université Libre de Bruxelles (ULB), Belgium. I would like to thank him for the inspiring discussions and his generous support during my visit. Collaborating with him during the year of my visit to ULB helped me to build one of the important cornerstones of my research and the thesis.

Further, I would like thank to all my colleagues and friends both at Aalborg University and Université Libre de Bruxelles. I am very grateful to have the chance to work at Database, Programming and Web Technologies (DPW) research group, which has been intellectually very stimulating and never boring. In particular, I would like to thank Kim Ahlstrøm M. Mathiassen from the very first year of my research, for being a helpful and enthusiastic colleague and a great friend. My special thanks go to İlkcan Keleş for his friendship and support, for our endless discussions and his valuable advice; I would also especially like to thank his wife Emel for her friendship and being with us during sleepless nights, while we were working. In addition, I would like to thank students, whom I co-authored research papers together: Alex B. Andersen, Kim Ahlstrøm, and Jacob Nielsen. Special thanks go to my significant other Mikael Midtgaard for writing the Danish abstract and co-authoring one of the journal papers during the Ph.D. studies. I am very grateful to Luis Galárraga for writing the French abstract within such a short

notice. I also would like to express my gratitude and appreciation to administrative staff Helle Schroll and Helle Westmark who have been always very helpful. In addition, I would like thank to my employer and colleagues at Accobat A/S, where I have been working as a BI consultant during the last two years of my Ph.D. studies, for their support. Especially, I would like to thank Mikael Iuel-Brockdorff and Jannick Raunow for accommodating and supporting me to complete this thesis.

Finally, I would like to thank my dearest friend Yasemin Gödel for always being there for me, believing in me, and supporting me mentally throughout writing this thesis and at every step of the Ph.D. studies. I would also like to thank Sofie Orry Amby for her friendship, support, and providing endless fun during the years in Aalborg and in Copenhagen. I thank my partner and my dearest - Mikael for comforting me and listening to me and helping me in every way he can. I am also very thankful to his family for their support. Last but not least, I would like to thank all my family members (my mom, my dad, my brothers Şamil and Münir and their wives and my friends Merve and Mine) for their unconditional love and support.

Nurefşan Gür
Aalborg, January 15, 2020

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence" - Doctoral College (IT4BI-DC).

Contents

Thesis Details

Thesis Title:	Modeling, Annotating, and Querying Geo-semantic Data Warehouses
Ph.D. Student:	Nurefşan Gür
Supervisors:	Prof. Torben Bach Pedersen, Aalborg University Prof. Katja Hose, Aalborg University Prof. Esteban Zimányi, Université Libre de Bruxelles

The main body of the thesis consists of the following papers.

[A] Alex B. Andersen, Nurefşan Gür, Katja Hose, Kim A. Jakobsen, and Torben Bach Pedersen. "Publishing Danish Agricultural Government Data as Semantic Web Data". In : *Semantic Technology - 4th Joint International Conference, JIST 2014, Chiang Mai, Thailand*, Vol. 8943, pp. 178-186, Springer LNCS, 2014.

[B] Nurefşan Gür, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. "Modeling and Querying Spatial Data Warehouses on the Semantic Web". In: *Semantic Technology - 5th Joint International Conference, JIST 2015, Yichang, China*, Vol. 9544, pp. 3–22, Springer LNCS, 2015.

[C] Nurefşan Gür, Torben Bach Pedersen, Esteban Zimányi, and Katja Hose. "A Foundation for Spatial Data Warehouses on the Semantic Web". In: *Semantic Web Journal*, Vol. 9, no 5, pp. 557–587, IOS Press, 2018.

[D] Nurefşan Gür, Jacob Nielsen, Katja Hose, and Torben Bach Pedersen. "GeoSemOLAP: Geospatial OLAP on the Semantic Web Made Easy". In: *Proceedings of the 26th International Conference on World Wide Web Companion (WWW 2017), Perth, Australia*, pp. 213–217, ACM, 2017. (*Awarded as **Best Demo** by Reviewer's Choice*)

[E] Nurefşan Gür, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. "Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP". In: *Semantic Technology - 6th Joint International Conference, JIST 2016, Singapore, Singapore*, Vol. 10055, pp. 287–304, Springer, LNCS, 2016.

[F] Nurefşan Gür, Torben Bach Pedersen, Katja Hose, and Mikael Midtgaard. "Multidimensional Enrichment of Spatial RDF Data for SOLAP". In preparation for submission to: *Semantic Web Journal*, 2020.

This thesis has been submitted for assessment in partial fulfilment of the PhD degree. The thesis is based on the submitted or published scientific papers, which are listed above. Parts of the papers are used directly or indirectly in the summary of the thesis. As a part of the assessment, co-author statements have been made available to the assessment committee and those are also available at the at the Technical Faculty of IT and Design at Aalborg University and the Faculty of Engineering at Université Libre De Bruxelles. The permission for using the published articles in the thesis has been obtained from the corresponding publishers with the conditions that they are cited and DOI pointers and/or copyrights/credits are placed prominently in the references.

Nurefşan Gür
Aalborg University, January 15, 2020

Part I

Thesis Summary

Thesis Summary

1 Introduction

1.1 Background and Motivation

The increasing popularity of Open Data continuously leads governmental organizations and agencies to share their content formally on the Semantic Web (SW) in many countries. Prevalence of public data on the SW made huge amounts of data accessible from different disciplines such as environment, health, agriculture, industry, statistics and geography in a non-proprietary open format: Resource Description Framework (RDF)[1]. From those disciplines, many of the data sets are containing spatial information with geographical coordinates. The geographical data on the SW needs to be treated differently as the spatial format of the data allows users to derive different perspectives with spatial analysis. Therefore, publishing and processing geographical/spatial RDF data have been very interesting research areas in the SW community. This increasing amount of spatial data from different disciplines on the geo-semantic web needs to be analyzed efficiently. In the non-semantic web database world, data warehouses are the best way to analyze relational data. Data warehouses require certain multi-dimensional (MD) modeling and analytical operators (**O**n-**L**ine **A**nalytical **P**rocessing - OLAP) for querying the data warehouse. Similarly, spatially extended MD models and spatial OLAP (SOLAP) operators are available in the traditional (non-semantic web) database world. Adapting these methods of building spatial data warehouses and querying spatial data warehouses for the SW, requires building SW ontologies/vocabularies for modeling spatial MD data, SOLAP query operators and techniques for supporting the SW query language: SPARQL, which stands for SPARQL Protocol and RDF Query Language. By following the SW standards and using the SW tools and technologies, we can truly structure, build and query geo-semantic data warehouses.

[1]RDF format allows data publishers to formulate statements about resources, where each statement consists of subject, predicate, and object, which compose a triple.

To build and publish geo-semantic data warehouses, firstly, it is essential to understand the geospatial semantic web and existing relevant research and technologies. Thus, as an initial effort the GovAgriBus Denmark RDF data set was published to the Linked Open Data (LOD) cloud (Fig. 1). The data set contains data from governmental, agricultural, business and geographical data. Geographical data contains spatial information, coordinates of the agricultural fields, which brings opportunities of interesting spatial queries (e.g., queries with spatial containment relationships or distance functions, etc.) across the data set. Examples of these queries could be that a customer demands in the market are for seasonal beer, Christmas trees, or pumpkins, where a supplier company would like to find specific and organic crops from a local region within North Jutland of Denmark. Such a query can be built by using the *within* containment relationship of organic fields of the crops in demand and the region border of North Jutland. The supplier company would also like to calculate the proximity of the organic field to its own location, where a spatial distance function can be utilized for calculating the distance between the location of the company and the field.

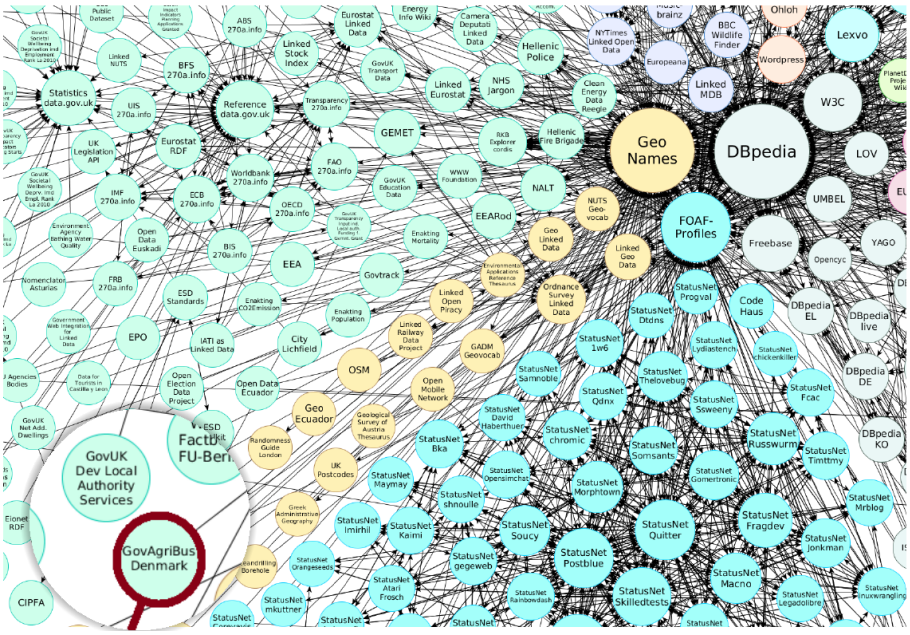


Fig. 1: GovAgriBus in LOD Cloud (2014)

As a general principle of publishing Linked Open Data, GovAgriBus data is linked with unique references (URIs) to external ontologies (such as GeoNames [36] and AGROVOC [31]), so that more related resources and things can be discovered via the URIs. Besides, this type of governmental data

(i.e., agricultural, environmental, geographical) is regularly published and updated from public portals (minimum frequency as a yearly basis), this brings the question on how to exploit this type of huge open geographic data with advanced analytical perspectives on the SW.

Our question led us to investigate how spatial data is handled in organizational data warehouses (DWs) with existing business intelligence (BI) technologies. Extending multi-dimensional (MD) models with spatial concepts and complex geometry features (i.e., continuous fields), defining an extended MD algebra that supports spatial data types, and extending On-Line Analytical Processing (OLAP) operations with spatial features are a few of the existing advancements in traditional spatial data warehouses. In order to query spatial data warehouses efficiently with advanced analytical queries, the concept of spatial OLAP (SOLAP) and spatially extended MD models have been widely implemented in *non-semantic* spatial data warehouses. Carrying these concepts to the geo-semantic web will be a significant improvement and key to answering our question.

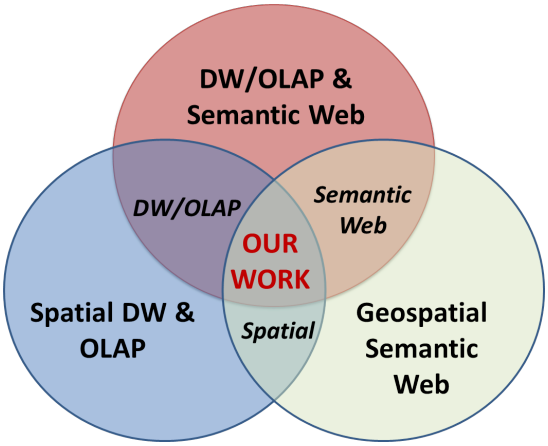


Fig. 2: Related work areas

The keyword (geo)*spatial* appears in the intersection of the above mentioned two research areas: Geospatial Semantic Web and Spatial DW/OLAP (Fig. 2). Before our attempt of implementing the geo-semantic data warehouse concepts, let us briefly ignore the *spatial* aspect of the two research areas and look into the intersection of DW/OLAP and SW to understand the current state of the BI on the SW. There are several approaches considered with RDF Data Cube (QB) Vocabulary[2] and QB4OLAP Vocabulary [10]. QB4OLAP addresses the limitations of QB vocabulary by introducing complex MD features such as dimensions with hierarchies, hierarchy steps and

[2]QB Implementations: `https://www.w3.org/2011/gld/wiki/Data_Cube_Implementations`

supporting OLAP queries directly on the RDF data. To support these advanced MD features on QB data sets, the QB2OLAP enrichment module is proposed [33] for a semi-automatic transformation of QB data sets with QB4OLAP semantics.

Even though there are promising research and technologies around data warehouses and OLAP on the SW, none of them supports spatial data warehouses or SOLAP operators over RDF data. Still, we use the existing technologies as a starting point for building geo-semantic data warehouses. This way, we acknowledge and utilize existing technologies and prevent redundant work. To build geo-semantic data warehouses we have studied the recent work thoroughly in the three main areas depicted in Fig. 2. Our work lays in the intersection of these three areas for building spatial data cubes and enabling SOLAP on the SW.

SOLAP operators utilize the geometry attributes of data warehouse members by including a spatial condition (such as a spatial function like closest *distance* or a spatial boolean predicate like *within*) in the analytical SOLAP query. Including spatial operations in a SOLAP query creates a dynamic interpretation of the data warehouse members, e.g., dynamic spatial hierarchy levels can be used to reveal new perspectives. For instance, a traditional OLAP operator to aggregate some numerical measures (e.g., population) from the *post-code area* level to *city* level does not require any spatial operations in the query. However, assuming that both the post-code area and city level members have geometry attributes representing their borders as a polygon geometry (with a set of coordinates), we would like to reveal new perspectives to calculate the total population in cities. Thus, we modify the query to aggregate the measures from post-code areas, which are not neighboring another city to the city spatial level. This means that we would like to calculate the population only in the inner cities, therefore, we have to use a spatial operation to exclude the postcode areas where the border of the area touches a city border, which can be done with SOLAP operators. Similar examples of SOLAP operators can be reproduced by calculating distance between the center points of the geometries.

Moreover, we can ensure the explicit annotation of topological relations between level members in geo-semantic data warehouses. For example, in a spatial hierarchy, some cities might belong to two geographical regions, meaning that the city borders intersect with two different geographical regions. This creates a *many-to-many* relation between *child-parent* (city-region) spatial levels in a data warehouse hierarchy, which might cause aggregating measures incorrectly between these levels. In a geo-semantic data warehouse, spatially enriched schema definitions allow us to explicitly annotate, when a city is *within* a region or when a city *intersects* a region. In order to prevent aggregating measures (e.g., population) incorrectly from city spatial level to region spatial level, we need first a spatial drill-down of measures to a lower

spatial level such as postal area, and then sum the total population of postal areas that are within the region.

By building geo-semantic data warehouses and addressing challenges such as querying with SOLAP operators or automated geo-semantic and MD annotation of existing RDF data sets, we aim to deliver a robust methodology with advanced technologies and tools, which are not available in the current state of the (geospatial) semantic web. Our approach aims at allowing BI and SW enthusiasts to query existing RDF data sets with SOLAP operators directly (from SPARQL endpoints), without needing to download the RDF data, model the offline data with spatial data warehouse concepts, and load to a spatial data warehouse to query with SOLAP operators, which is non-practical, isolated, and fundamentally against to the rationale of semantic web and linked open data principles.

We now give an overview of the contributions of the thesis, with further details given in the next Sections 2 - 6.

1.2 Geo-semantic Data Warehouses and Spatial OLAP

We propose a cube (QB) vocabulary for spatial OLAP on the Semantic Web - QB4SOLAP, which is a generic and extensible vocabulary, suitable for MD modeling of all kinds of geographical domain data sets to publish on the SW. QB4SOLAP is based on the latest stable version of the QB4OLAP Vocabulary. We validate QB4SOLAP by *modeling* two non-trivial use cases with QB4SOLAP, which have complex geometries and spatial MD concepts such as spatial hierarchy steps (between spatial dimension levels) that require annotation of topological relations between the level members.

Together with QB4SOLAP, common SOLAP operators are formally defined with MD semantics over RDF data. Previously modeled use cases are published as geo-semantic data warehouses in RDF format, where we can test the SOLAP operators. When a SOLAP operator is written in SPARQL, it can be very complicated for new BI users on the SW. As a common BI practice, OLAP/SOLAP queries are written in nested form, which makes writing nested SOLAP queries in SPARQL extremely long and overwhelming for inexperienced users. Thus, we have a outlined vision of geo-semantic data warehouses with a set of frameworks and tools that can address these and similar challenges (Fig. 3).

In this line of work, we provide algorithms for translating individual or nested SOLAP operators to SPARQL queries by using the spatial MD model semantics and formal semantics of defined SOLAP operators. On top of these algorithms, the GeoSemOLAP tool is implemented for users to formulate SOLAP queries interactively via using a graphical user interface and maps, which are instantly translated to SPARQL.

In order to fully utilize the potential of the semantic web, existing RDF

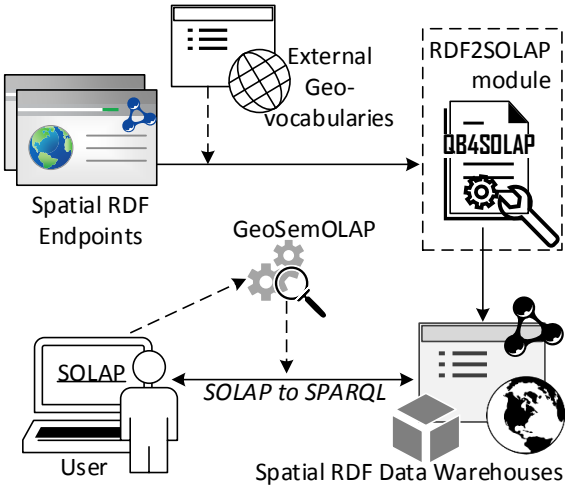


Fig. 3: Geo-Semantic Data Warehouses Vision - Adapted from [15]

data sets that are also suitable for spatial and MD modeling (e.g., QB4OLAP RDF data sets with level attributes that have geographical data) are enriched in an automated way with QB4SOLAP semantics. This is achieved by the RDF2SOLAP enrichment module. RDF2SOLAP defines two families of algorithms for hierarchical enrichment and factual enrichment. In each family, there are algorithms for 1) utilizing existing hypothetical QB4OLAP roll-up relations to detect spatial relations between level members and level members and fact members 2) assuming there are no explicit roll-up relations and discovering the spatial relations between all members.

1.3 Organization

The rest of the thesis summary is organized as follows. Section 2 summarizes Paper A, which is an initial attempt to publish Danish governmental and spatial data set as Linked Open Data in order to derive new spatial relations and analytics insights with SPARQL aggregate queries. Section 3 presents a detailed overview of Paper B and Paper C, where the QB4SOLAP Vocabulary is introduced with multi-dimensional and spatial RDF semantics, SOLAP operators are explained with high-level semantics and SOLAP to SPARQL generator algorithms are given with examples. Section 4 gives an overview of Paper D and explains the implementation and workflow of the GeoSemOLAP tool, which is implemented as a tool using SOLAP generator algorithms. A non-trivial nested set of SOLAP operations is demonstrated with GeoSemOLAP in Section 4. Section 5 summarizes Paper E with

a real-world QB4SOLAP use case from Danish governmental, environmental, and geographical data sets together with interesting SOLAP operations and new analytical perspectives in SPARQL. Section 6 summarizes Paper F, introduces an RDF2SOLAP enrichment framework and outlines the spatial multi-dimensional enrichment process for various cases with enrichment algorithms. Finally, Section 7 gives a summary of the contributions and Section 8 concludes the thesis summary and presents directions of future work.

2 Publishing Spatial and Governmental Linked Open Data

This section gives an overview of Paper A [3].

2.1 Motivation and Problem Statement

The increasing popularity of publishing Open Data on the Semantic Web and Linked Open Data(LOD)[3] movement attracted many public, governmental and non-governmental organizations. The Linked Open Data movement encourages data providers to publish and connect distributed data across the web by following a list of best practices [19] and using common web standards (i.e., RDF, SPARQL, and HTTP URIs). Using these standards while publishing Open Data on the web ensures discoverability, makes the data easily dereferenceable (via URIs), and makes it relatable to other available public resources.

Especially for governments, the main goal of publishing Open Data is to improve collaboration, inspire innovative applications, and also attract entrepreneurs, which can lead to growth in the prosperity of the country and resourcefulness of the governments, besides providing transparency to the public at the governmental data level.

The Danish government has also started publishing Open Data on the web at the end of 2012 [5], where several ministries and governmental agencies from various domains (e.g., agriculture, forestry, and fishery, statistics, geographical and environmental domains) made their data publicly available. However, the publishers made their raw data available in heterogeneous formats such as PDF, CSV, XML, SHP (for geospatial data), XLS, etc. but none of these data were available on the Semantic Web as Linked Open Data.

Our primary goal is to publish Danish governmental Open Data (from a selected range of non-trivial domains) on the Semantic Web by following the best practices of publishing Linked Open Data. We also assess the challenges and share our experiences in order to provide guidelines for publishing Danish governmental data on the Semantic Web.

[3] Linked Open Data (LOD) Cloud: `http://lod-cloud.net/`

2.2 Use Case Data Lifecyle

We primarily choose to focus on the agricultural domain, since it is a non-trivial and an interesting use case. It is non-trivial because agricultural fields and areas have spatial information, thus, we encounter the complexity of the spatial data format, besides we can link the spatial information with external geographical data. It is interesting because the majority of Denmark's land use is spared to agriculture[4] with 60% - 70. Secondarily, we decided to combine the agricultural data with company data, which can provide interesting query patterns for the agricultural industry that are not readily available by considering the data sets separately.

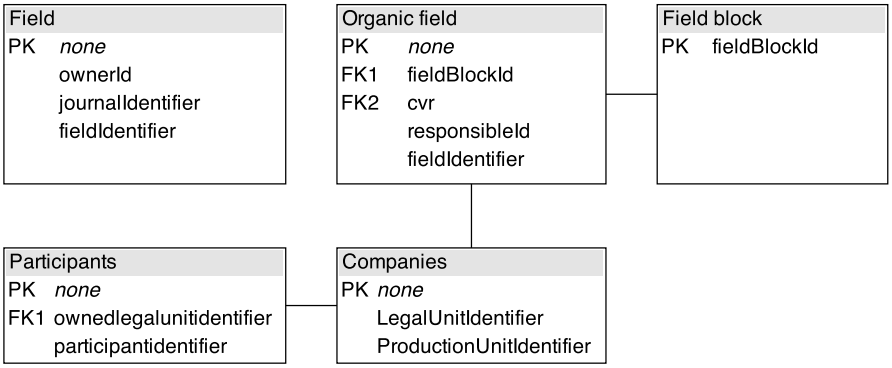


Fig. 4: Relational schema of the use case data sets (adapted from [4])

GovAgriBusDenmark Raw Data sets

Linked Open Data set of GovAgriBusDenmark use case is composed of Danish **Gov**ernmental, **Agri**cultural, and **Bus**iness data sets. Initial raw data for governmental and agricultural data is published by the Ministry of Food, Agriculture, and Fisheries of Denmark (FVM) in geospatial data (SHP) format. Business data is published by Central Company Register (CVR) in CSV format.

We give an overview of the relational database schema of the use case data sets from agricultural and business domains in Fig. 4. Agricultural data has three main data sets in SHP format (with POLYGON coordinates): Field, Organic field, and Field block. Organic field and Field block data sets have referential integrity through the *fieldBlockId* key, which can be easily related and linked. Field and Organic fields are related by using spatial joins (computed from the spatial coordinates) through a GIS tool.

[4]https://tradingeconomics.com/denmark/agricultural-land-percent-of-land-area-wb-data.html

The business data has two main data sets in CSV format: Company and Participant. These data sets have referential integrity through the *LegalUnitIdentifier*, which is the same as a unique CVR number for the companies. A company is constituted by legal units and production units. Every legal unit has at least one production unit. The relationship between legal units and production units is illustrated in Fig. 5. Participants can be a person with a unique Danish social security number (CPR number) or a legal unit with a CVR number.

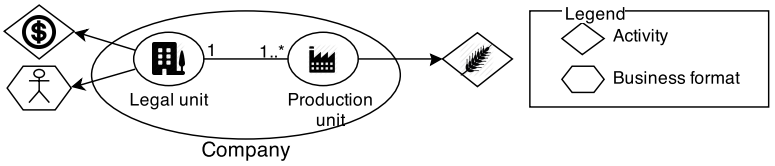


Fig. 5: Company structure (adapted from [4])

In Table 1, we give an overview of the number of records and attributes from the raw (tabular) data sets from the selected use case domains.

Table 1: Use Case Raw Data Profile

Domain	Data set		Number of columns (attributes)	Number of rows (records)
Gov. Agri.	Field		9	641,081
	Organic Field		12	52,060
	Field Block		12	314,648
Bus.	Company	Legal Unit	59	>600,000
		Production Unit	59	650,000
	Participant		7	350,000

Data annotation and reconciliation

We follow an iterative transformation and integration process of: *importing*, *analyzing*, *refining*, *linking* and *publishing* of data. The details of these main activities are as follows:

Import. The first steps are to extract the raw data from the public resources in its original format (SHP and CSV) and import it to a common operational environment such as a relational database. At this step, for spatial data sets, we initially import them to a GIS tool in order to implement spatial joins and associate disjoint spatial data sets with referential integrity before loading into a relational database.

Analyze. During data analysis, we profile the data sets, collect statistics and get a better understanding of the attributes (columns), in order to formalize an ontology for data annotation. During analysis, we decide on re-using the existing ontologies in relation to our research domain as give in the following.

- **WGS84:** Used for defining the spatial objects/coordinates [7].
- **GeoNames:** Similarly, used for defining spatial objects and external linking of place names (municipality names) [36].
- **AGROVOC:** Used for defnining fields and crops by linking with [31].
- **FOAF:** Used in business data `rdfs:subClassOf-foaf:organization` [1].

Refine The refinement step is the last part before linking and publishing the data, where we get rid of the corrupt data, inconsistent fields, etc. by cleansing and conversion scripts.

Link and Publish. We generate mappings by using Virtuoso Open Source [29], where RDF data is generated and directly published.

We distinguish two different linking methods: Internal linking and external linking. Internal linking is done, while we create our GovAgriBusDenmark ontology at the schema level, where we re-use external vocabularies. Internal linking is at the conceptual level where we define the relationships between classes and concepts that were identified during the Analyze stage.

> **Example 1 (Internal Linking)**
> The following example represents the internal linking of the Field and Field Block classes using the `geonames:contains` predicate [3].

```
agri:contains rdf:type owl:ObjectProperty ;
    rdfs:domain agri:FieldBlock;
    rdf:range agri:Field ;
    rdfs:subPropertyOf geonames:contains .
```

External linking is implemented at the instance level, where we link e.g., the URIs of the place names to their equivalent geographical names from external ontologies and gazetteers.

> **Example 2 (External Linking)**
> The following example represents external linking of place name Aalborg municipality (from the business data set) to GeoNames URI of Aalborg with`owl:sameAs` predicate.

```
bus:Aalborg rdf:type bus:municipality ;
    wgs:lat "57.048"xsd:float ;
    wgs:long "9.9187"xsd:float ;
    owl:sameAs geonames:26224886 .
```

In total 32.457,657 triples are generated and published via SPARQL endpoint `http://extbi.lab.aau.dk/sparql`. The final Linked Open Data Sets and ontologies are also published to the LOD cloud and available for download from datahub.io[5].

2.3 Technical Details and Query Runtimes

Finally, we present our system architecture and the set-up of experiments. The system architecture is given in Fig. 6. The source layer is located on various machines of the authors, and the hardware set-up of the source layer is not relevant for (native) RDF query run-times. Hardware set-up of the transformation and conceptual layer is an Ubuntu 13.10 Saucy server with 3.4 GHz Intel Core i7-2600 processor and 8 GB RAM, running OpenLink Virtuoso 07.00.3203.

Import and Analyze (partially Refine) stages are hosted on the source layer. In the transformation layer, we generate the RDB to RDF mappings by using Quad (Map) Patterns. In order to map the raw literal data values (float, string, etc.) IRI classes are used. The conceptual layer receives queries from the presentation layer sent by the end-users. If the queries are already cached, they are answered directly from the LOD repository cache, otherwise, they are passed to the Native RDF Quad Store. If a query can be answered by the quad store, Virtuoso provides the ontologies to annotate the data and rewrite the query using inference in the given ontology [4].

Experiments

We have described three materialization strategies: *virtual*, *relational materialization*, and *native* RDF. Figure 7 shows different data flows for these strategies with solid lines for processing the integration and load of the data, with dashed lines for processing the queries.

The *virtual* strategy follows the arrows 1, 2 and 3 during query processing (Fig. 7). We cleanse the data in relational SQL views and map to RDF data from these views. In order to optimize the performance, indexes on keys and spatial attributes are created. The *relational Materialization* strategy is represented through the arrows: 4 during load, and 3 and 5 during query processing. Relational tables in this strategy are created from the previous SQL views with similar indexes. Finally, the *Native RDF* strategy is created

[5] `https://datahub.io/dataset/govagribus-denmark`

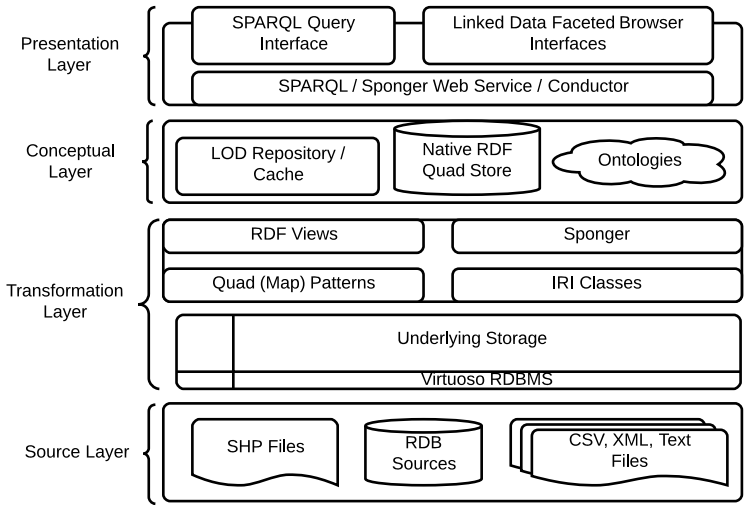


Fig. 6: Multi-layered system architecture (adapted from [4])

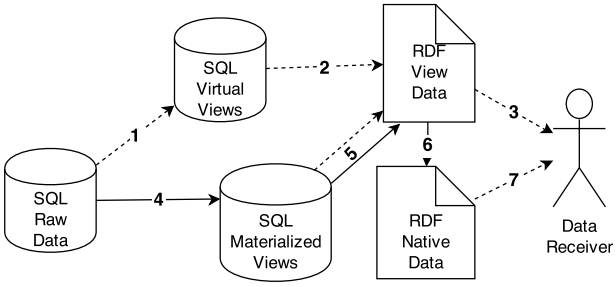


Fig. 7: Data flow for the materialization strategies (adapted from [3])

by following the arrows 4, 5, and 6 during load and 7 during query processing. This strategy is run natively on RDF data in a triple store. The RDF data is created from the materialized views by annotating with the selected ontologies and mapping them into the triple format.

Tables 2 and 3 show the processing times for data loading/transformation and running the queries. We provided query run times for a number of query templates, where **AQT** stands for *aggregated query templates* and **SQT** stands for *standard query templates*. Aggregated query templates use aggregate functions (e.g., MAX, SUM, COUNT, etc.) with grouping, where standard query templates use basic transactional query constructs and SPARQL1.0 constructs[6].

[6]Supporting aggregate queries on SPARQL started in March,2013 with the release of

Table 2: Load times in seconds (adapted from [3])

Step	Virt.	Rel.	Nat.
Data			
Cleansing	74.92	603.35	603.35 5
Load			
Ontology	1.01	1.01	1.01
Load			
Mappings	8.76	12.35	12.35
Dump			
RDF	0.00	0.00	4684.82
Load			
RDF	0.00	0.00	840.04
Total	84.68	616.70	6141.56

Table 3: Query runtimes in seconds (adapted from [3])

Query	Virt.	Rel.	Nat.
AQT 1	5.92	3.39	1.04
AQT 2	13.32	7.00	0.23
AQT 3	10.81	7.70	0.05
AQT 4	–	–	0.14
AQT 5	–	20.37	0.86
SQT 1	–	–	2.35
SQT 2	0.09	0.12	0.10
SQT 3	2188.85	1.81	0.40
SQT 4	6.57	2.35	1.63
SQT 5	–	23.79	3.29
Average	370.93	8.31	1.01

By looking at the processing times, we have concluded that the virtual strategy has the fastest load times (Table 2), however, it is the least efficient in answering the queries. The native RDF strategy, on the other hand, is the most efficient in answering even the aggregated queries fast, while it demands more time during preparation and load of the RDF data as expected. The relational strategy can be seen as a trade-off between load times and query processing times, it is ten times faster in load times and eight times slower in query run times than the native RDF strategy. For rapidly changing data, where loading the data on a frequent basis is required, the virtual or the relational strategy would be the most fitted. On the other hand, for large volumes of data where aggregated querying is also required, the relational strategy or the native RDF strategy would be most efficient by looking at the query run times.

2.4 Conclusion and Discussion

Prompted by the popularity of the Open Data movement and the opportunities that the Semantic Web can provide for governmental data publishers, we have investigated how to publish Danish governmental data sets (from agricultural domain) on the Semantic Web. Furthermore, we have shown how to link this domain with business data and external sources (GeoNames), and annotate by using existing vocabularies.

In order to present the best practices, we have tested different extract, transform, and load (ETL) strategies for preparing the Semantic Web data in RDF format from heterogeneous sources in several formats. We have selected the governmental agricultural domain with data sets in SHP file for-

SPARQL1.1 – https://www.w3.org/TR/sparql11-query/

mat and business domain with data sets in CSV file format. The large part of our implementation and the system architecture is based on Openlink Virtuoso, where both native and RDBMS-based RDF storage is supported. We have reconciled the use case data from SQL Server views/tables to Virtuoso RDF store graphs. We have mainly used SQL views and scripts for data cleansing besides we have used other third-party tools such as OpenRefine[7] (formerly known as Google Refine) for data cleanup and transformation. In order to map the relational data to RDF, we have initially tried third party ontology mapping (RDB2RDF) tools such as D2RQ[8], however, we eventually decided to proceed with a built-in RDFizer middleware Virtuoso Sponger, which semi-automatically extracts the source data from various formats and maps to RDF directly on Virtuoso server (Fig. 6). In order to annotate the use case data, we have developed a new ontology, where we reused existing ontologies. An interesting challenge we have encountered was deriving spatial containment relationships between field and field block data that were not encoded in the original datasets (Fig. 4), where we had to implement spatial joins using a third party GIS tool.

Next, we have presented our experiences with different load times and query processing scenarios. The native RDF strategy demonstrated the most efficient query run times compared to the two other strategies (relational and virtual strategies), however, the native strategy decouples the RDF data from the relational data, thus it is well suited for static data looking at the expensive load times. The native RDF strategy requires technical improvements to provide more efficient transform and load times.

The RDF data and all our experiments with a number of aggregated and standard query templates (with and without built-in spatial functions) are provided through our project web site[9].

The work demonstrated within the scope of this paper is our first small step to create "Geo-semantic Data Warehouses". We have examined the available data sets from different domains with non-trivial spatial data, then, we published RDF data from the agricultural domain. As proof of concept, we measured query performance for aggregated query templates, which are fundamental in data warehousing and multi-dimensional models. We acquired promising results from our experiments for AQTs that run on native RDF data. The next step is to get closer to creating Geo-semantic Data Warehouses is to define conceptual models/vocabularies for annotating spatial multi-dimensional data and show how we can make a difference with spatial multi-dimensional query templates as explained in Section 3.

[7] http://openrefine.org/
[8] https://www.w3.org/2001/sw/wiki/D2RQ
[9] http://extbi.cs.aau.dk/govAgriBus/

3 Enabling SOLAP operations in SPARQL over Spatial Multidimensional Data Cubes on the SW with QB4SOLAP Vocabulary

This section gives an overview of Paper B [12] and Paper C [15]. Paper B is a subset of Paper C. Paper C is extended from Paper B as a journal paper, where the semantics of MD Data cubes with QB4SOLAP are improved and formal definitions of SOLAP operators are revised. Moreover, we have provided algorithms in Paper C for generating SPARQL queries from high-level SOLAP operators.

3.1 Motivation and Problem Statement

Paper A demonstrated a use case on, how to publish Danish Agricultural and Business data on the Semantic Web (SW). In the experiments of Paper A, it is shown that aggregated queries demonstrated good performance on the Semantic Web for a non-trivial set of domain data sets with complex data types. The advances in the Semantic Web technologies make it possible to query Linked Open Data with On-Line Analytical Processing (OLAP) style queries containing aggregate operators. OLAP relies on multi-dimensional (MD) data models and schemas used in Data Warehouse (DW) systems.

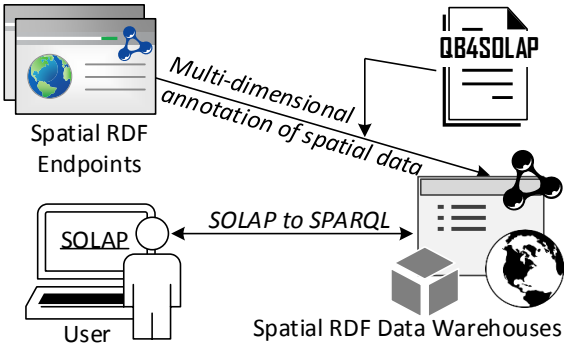


Fig. 8: QB4SOLAP approach to SOLAP on the SW (adapted from [15])

The increasing popularity of publishing LOD data sets from various domains (e.g., environmental, agricultural, climate and statistical data) with spatial information on the SW brings opportunities for advanced (spatial and multi-dimensional) analysis of these data sets. Several interesting spatial RDF

endpoints are published to the LOD cloud [15][10,11,12,13]. We can reveal interesting results for decision-makers and domain experts by querying these data sets with analytical queries and spatial OLAP (SOLAP), however, the data sets should be modeled and annotated with spatial and MD concepts that are fundamental for SOLAP operations, beforehand being published on the SW. The existing SW technologies only support annotating *non-spatial* multi-dimensional data, which limits online analytical processing (OLAP) queries over the non-spatial SW data sets. Fig. 8 shows a general overview of our approach for enabling SOLAP and spatial multi-dimensional data warehouses (a.k.a data cubes) on the Semantic Web. In the current state of the SW, users and decision-makers cannot query spatial RDF endpoints with SOLAP operations, unless the RDF data is downloaded, mapped to a relational data model e.g., star schema and imported to a traditional data warehouse that supports spatial functions. This is a labor-intensive and cumbersome process, which eventually isolates the published linked open data in a non-open format in data silos and that defeats the purpose of publishing linked open data in the first place. We propose QB4SOLAP - a vocabulary for annotating spatial multi-dimensional data on the Semantic Web. Thus, the users can query the Semantic Web data with SOLAP operations.

3.2 State of the Art

The following research areas are a recap from Fig. 2 with related work references in each area.

DW/OLAP on the SW. Practice of multi-dimensional (MD) data warehouses (DW) with RDF data and performing OLAP operations on the RDF data in SPARQL have been an interesting research topic by many other researchers [10, 18, 35]. However, none of the researchers focus on *spatial* data warehouses or *spatial* OLAP (SOLAP) operations on the Semantic Web. The state of the art of multi-dimensional modeling and analysis on the SW is restricted to *non-spatial* Semantic Web data.

Spatial DW/SOLAP. In a non-SW context, staging spatial data in data warehouses and querying spatial data warehouses with extended OLAP operations with spatial functions is a prevalent research field. The term spatial OLAP (SOLAP) was first introduced in 1997 [6] for spatial data warehouses and many researchers since then have been following up with valuable contributions to the community [8, 20, 26].

[10]EuroStat: `http://ec.europa.eu/eurostat`
[11]UK Environmental Data: `http://environment.data.gov.uk`
[12]Danish Agricultural Data: `https://datahub.io/dataset/govagribus-denmark`
[13]Australian Climate Observations: `http://datahub.io/dataset/acorn-sat`

Traditional DWs annotate the location/place dimension as a conventional steady dimension without spatial features or attributes (e.g., coordinates) but with alphanumeric attributes (e.g., a nominal reference to a place name). Similarly, measures do not have spatial information such as coordinates. Therefore, it is not possible to carry out advanced spatial analysis such as deriving *topological relations* between *hierarchy levels* or define and operate with *spatial aggregate functions* on spatial *measures* in non-spatial DWs [15]. In spatial data warehouses, through the geographical coordinates of the location data, we can also derive new perspectives (e.g., dynamic spatial hierarchies) during query run time.

Geospatial SW. Semantic Web accommodates large volumes of geospatial data, with the increasing popularity of publishing Linked Open Data by several organizations. Various projects and organizations publish spatial data on the Semantic Web (e.g., LinkedGeoData [30] interactively transforms OpenStreetMap to RDF data, GeoKnow [28] links geospatial data from heterogeneous resources on the SW), as we have also published Danish agricultural and business data (with spatial coordinates) to the LOD cloud as GovAgriBus in Paper A (Section 2).

Spatial DW/SOLAP on the SW (OUR WORK). We have investigated the state of the art for the three distinct highlighted research areas given above. Our rationale for enabling spatial OLAP on the Semantic Web lays in the intersection of these three areas.

Carrying spatial data warehouses and SOLAP to the SW context can provide methods for advanced analysis of geospatial data on the SW and easily improves the existing models and tools for a thorough analysis of multidimensional data with spatial support.

3.3 Spatial Multidimensional Data Semantics

The main fundamentals of spatial data warehouse (a.k.a spatial cube) concepts should be defined precisely with an explicit vocabulary on the SW in order to benefit from the spatial analysis capabilities of SOLAP operators. During our survey on the state of the art of *DW/OLAP on the SW*, we found QB4OLAP vocabulary [10], the most prominent in structure and technical advances to support modeling and annotating (non-spatial) data warehouses on the Semantic Web. QB4OLAP is extended from RDF Data Cube (QB) Vocabulary[14], and supports OLAP operations with a full-bodied MD metamodel, while QB vocabulary is limited to statistical data models without OLAP di-

[14]RDF Data Cube (QB): https://www.w3.org/TR/vocab-data-cube/

mensions and hierarchies. As a starting point, we decided to propose a spatially extended metamodel on top of QB4OLAP vocabulary - QB4SOLAP.

QB4SOLAP Vocabulary

Version History. During the design of QB4SOLAP Vocabulary, there was an earlier (non-stable) version of the QB4OLAP vocabulary(v1.1) available under development. Our initial design (QB4SOLAPV1.1) was based on this version of QB4OLAP, which is later stabilized in version v1.2 [15]. In Paper B we have published QB4SOLAP V1.2 (Fig. B.3) based on the latest stable version of QB4OLAP(v1.2) at the time being. With minor revisions of RDF instance and RDF class representations from V1.2 to V1.3, the latest version of the QB4SOLAP Vocabulary is published with Paper C, which is given in Figure 9 [15]. All versions of QB4SOLAP is available from our project website [16].

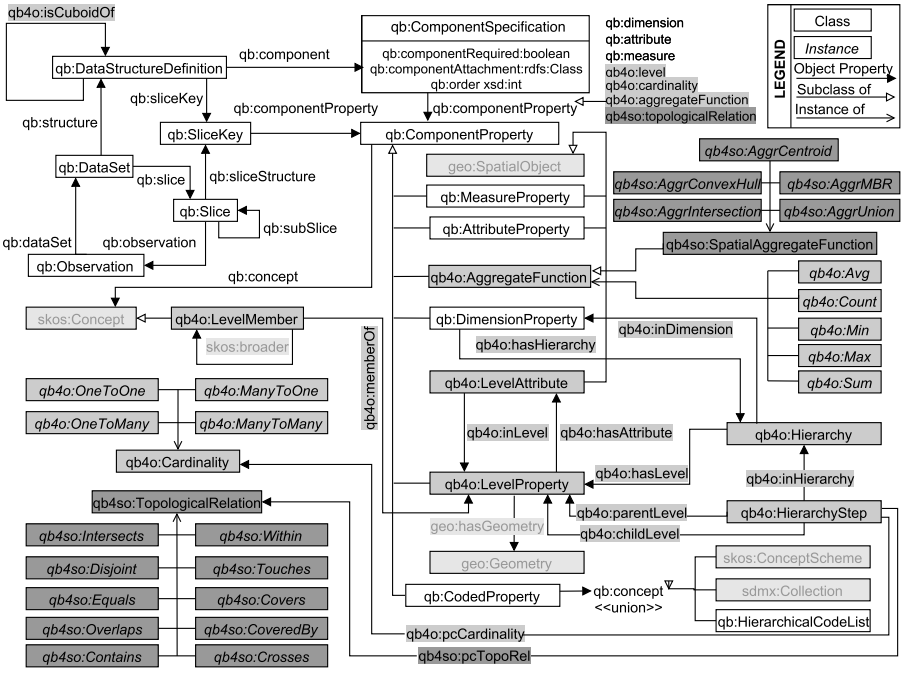


Fig. 9: The QB4SOLAP vocabulary (V1.3) (adapted from [15])

Multidimensional Data Cube. QB4OLAP vocabulary defines all the concepts from multi-dimensional models in RDF terms. Multidimensional mod-

[15] QB4OLAP: `http://purl.org/qb4olap/cubes_v1.2`
[16] QB4SOLAP: `http://extbi.cs.aau.dk/QB4SOLAP/index.php#qb4solapversions`

els are usually interpreted as *data cubes* [12, 15], where cells of the cube correspond to *observation facts*, which are the center of MD analysis. Facts have a set of attributes known as *measures*, with defined *aggragate functions*. On a multi-dimensional space, data cube has *n dimensions* (with contextual information) to provide different analysis perspectives, where each dimension can have *hierarchies* with *levels*. Levels (of dimensions) allow users to aggregate measures (of facts) at different levels of detail (a.k.a. granularity). In order to support this kind of analysis at different granularities, dimensions and facts are related in the structure and definition of a data cube. Level have also *attributes*, which define the basic characteristics of the level members[17].

We distinguish the MD data cube elements by two definitions levels for defining and annotating with QB4SOLAP, at the schema level and the instance level, respectively. Cube elements such as dimensions, hierarchies, hierarchy steps, levels, attributes, fact observations, measures, etc. are the *schema level* cube elements. By using the QB4SOLAP vocabulary these MD concepts (cube elements) and their roles and relations to each other are annotated at the schema level in RDF triple format. From a relational point of view, schema level (RDF) cube elements correspond to tables and columns (of tables) in an MD Star Schema model. Fact members with measure values, level members with attribute values are *instance level* MD concepts, where we annotate them by using QB4SOLAP vocabulary at the instance level. These instance level (RDF) cube elements correspond to the actual data rows/records, which are annotated with QB4SOLAP in triple format. In Fig. 10, an MD conceptual schema of a sample use case - GeoNorthwind [18], which is used in [12, 15] is given.

In Paper B, we give the formal RDF semantics of each MD concept (abovementioned cube elements) and spatial extensions (Sections 4.1 and 4.2), with respect to the common definition of an RDF triple[19]. In Paper C, we have improved the presentation and explanation of formal definitions and spatial extensions (Sections 4.1 and 4.2). The following definition (Def. 1) for *Hierarchy steps* is reproduced from Paper C. An overview of a hierarchy step (at the schema level) - `qb4o:HierarchyStep` is depicted in Fig. 9 (bottom right) .

Definition 1. (Hierarchy steps - Reproduced from [15]) A hierarchy h has a set of hierarchy steps $HS(h) = \{hs_1, \ldots, hs_q\}$, which define the structure of the hierarchy in relation with its corresponding levels. A hierarchy step

[17] Both Papers B [12] and C [15] give this general definition of MD cube concepts with examples briefly in the introduction sections

[18] GeoNorthwind Datawarehouse is an extended version of Northwind database with spatial data and modeled in a multi-dimensional way. Northwind Database: https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs

[19] An *RDF* triple t consists of three components; s is the subject, p is the predicate, and o is the object, which is defined as: $triple(s, p, o) \in t = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \times \mathcal{L})$ where the set of IRIs is $\mathcal{I}$, the set of blank nodes is $\mathcal{B}$, and the set of literals is $\mathcal{L}$.

$hs_i = (l_c, l_p, card) \in HS(h)$ entails a roll-up relation between a lower (child) level l_c to an upper (parent) level l_p with a cardinality *card*. The cardinality $card \in \{1\text{-}1, 1\text{-}n, n\text{-}1, n\text{-}n\}$ describes the number of members in one level that can be related to a member in the other level for both the child and the parent levels.

Each hierarchy step hs_i is defined in the cube schema graph $\mathcal{G}^S$ as a blank node `_:hs`$_i \in \mathcal{B}$ with the `qb4o:HierarchyStep` predicate. Each hierarchy step is linked to its hierarchy with the `qb4o:inHierarchy` property. The child and parent levels are linked in a hierarchy step with the `qb4o:childLevel` and `qb4o:parentLevel` properties, respectively. The cardinality *card* of a hierarchy step is defined by the `qb4o:pcCardinality` property. The RDF graph formulation of the hierarchy steps $HS(h)$ is represented as

$$\mathcal{G}^S_{HS(h)} = \bigcup_{i=1}^{q} \mathcal{G}^S_{hs_i}$$

where

$$\begin{aligned}
\mathcal{G}^S_{hs_i} = \\
&\{(\texttt{_:hs}_i\ \texttt{rdf:type}\ \texttt{qb4o:HierarchyStep})\} \cup \\
&\{(\texttt{_:hs}_i\ \texttt{qb4o:inHierarchy}\ id^S(h))\} \cup \\
&\{(\texttt{_:hs}_i\ \texttt{qb4o:parentLevel}\ id^S(l_p))\} \cup \\
&\{(\texttt{_:hs}_i\ \texttt{qb4o:childLevel}\ id^S(l_c))\} \cup \\
&\{(\texttt{_:hs}_i\ \texttt{qb4o:pcCardinality}\ id^S(card))\}
\end{aligned}$$

Spatial Extensions. Our spatial extension in QB4SOLAP is inherited from GeoSPARQL[20] definitions, where they are supported for topological relations and spatial aggregate functions.

In Fig. 9, on the left bottom corner, *topological relations* are given as a spatial extension to the QB4OLAP vocabulary hierarchy steps. Hierarchy steps occur in a hierarchy, where they relate the levels of a hierarchy from a lower (child) level to a higher (parent) level. If a level has (a spatial) geometry, which means that the spatial coordinates of the level members are recorded as geometry data types at the instance level (directly on the data).Therefore, the topological relations between the (parent-child) level members can be derived and annotated both at the schema level and instance level.

On the top-right corner of Fig. 9, *spatial aggregate functions* are given as subclass of QB4OLAP aggregate functions. The spatial extensions use `qb4so:` prefix. By using these extended schema level definitions, we can annotate any MD data cube with QB4SOLAP in RDF format. Spatial aggregate functions

[20]GeoSPARQL Ontology (`http://www.opengis.net/ont/geosparql#`) is an Open Geospatial Consortium (OGC) standard for annotating spatial data and functions in RDF.

are defined over spatial measures, where a measure should have a geometry data type with coordinates at the instance level.

By using these spatial extensions from the QB4SOLAP vocabulary, we can re-define a hierarchy step as *spatial* hierarchy step with the spatial extensions. The following extension (Ext. 1) for *spatial* hierarchy steps is reproduced from Paper C. The spatial hierarchy steps formalization uses the topological relations given in Fig. 9 (bottom left).

Extension 1. (Spatial hierarchy steps - Reproduced from [15]) A hierarchy step is spatial if it relates a spatial child level l_{c_s} and a spatial parent level l_{p_s}, in which case it entails a topological relationships between these spatial levels. A spatial hierarchy step is then a tuple $hs_{i_s} = (l_{c_s}, l_{p_s}, card, topoRel)$ where the topological relation $topoRel$ belongs to the $\mathcal{T}_{rel}$ class (Def. 2). The topological relation between parent-child levels of a spatial hierarchy step is defined by the `qb4so:pcTopoRel` property. The RDF graph formulation of the spatial hierarchy steps $HS_s(h)$ (w.r.t. Def. 9) is represented as

$$\mathcal{G}^S_{HS_s(h)} = \bigcup_{i=1}^{q} \mathcal{G}^S_{hs_{i_s}}$$

where

$$\mathcal{G}^S_{hs_{i_s}} = \mathcal{G}^S_{hs_i} \cup \{(\_{:}\texttt{hs}_i \ \texttt{qb4so:pcTopoRel} \ id^S(topoRel))\}$$

Example 3 (Spatial Hierarchy Step Example - Reproduced from [15])
The triples below show how the hierarchy steps of the Geography spatial hierarchy in the Customer dimension of the GeoNorthwind DW (Fig. 10) are represented in RDF using Def. 1 and Ext. 1. Note that all hierarchy steps are spatial and have an associated topological relation `qb4so:Within`. These topological relations in the conceptual schema figure are denoted at each spatial hierarchy step with white in black circle signs.

```
# Hierarchy steps
_:customerGeography_hs1 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:customer ;
   qb4o:parentLevel gnw:city ;
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so:Within .
_:customerGeography_hs2 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:city ;
   qb4o:parentLevel gnw:state ;
```

```
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so:Within .
_:customerGeography_hs3 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:state ;
   qb4o:parentLevel gnw:country ;
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so:Within .
```

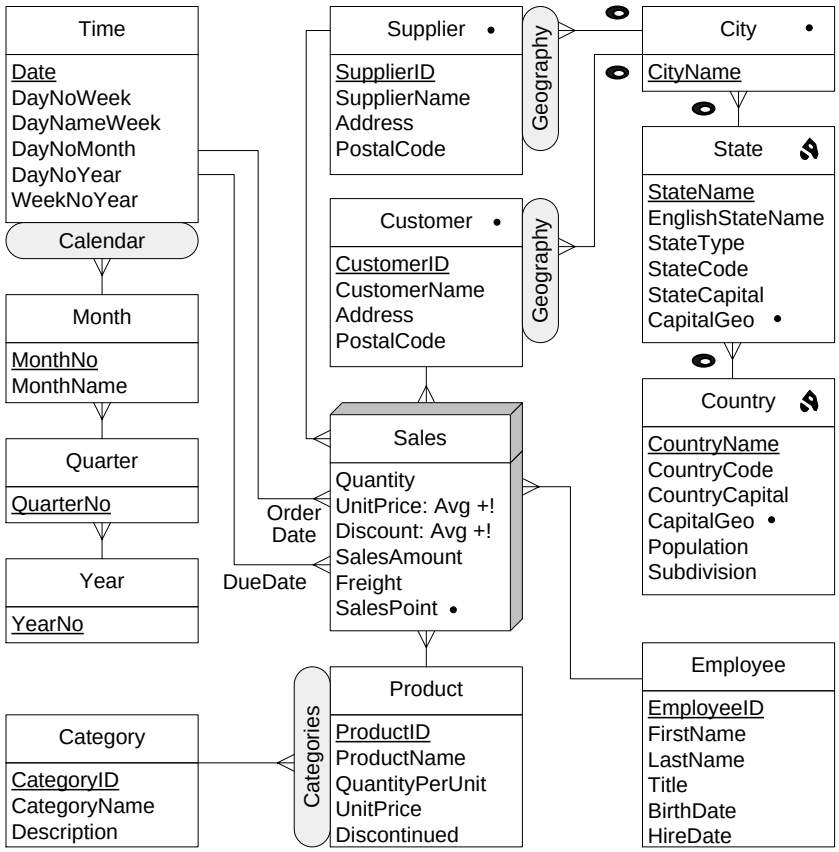


Fig. 10: Conceptual multi-dimensional schema of the use case data: GeoNorthwind data warehouse: The center of analysis is *Sales fact* cube in the middle. *Measures* are listed within the fact cube. Fork shaped arrows link *dimensions* to the fact through the base *levels*. Level *attributes* are listed within each level. *Hierarchies* are given in ellipse boxes attached to the base level of the dimension. Spatial levels and attributes are given with point and polygon signs. Spatial *hierarchy steps* are denoted with a sign (white in black circle) to represent the *within* relations (adapted from [15])

SOLAP Operations

We have grouped the spatial operations into three classes in Paper B; *spatial aggregation* $\mathcal{S}_{agg}$, *topological relation* $\mathcal{T}_{rel}$ and *numeric operation* $\mathcal{N}_{op}$ [12].

SOLAP operations are built on top of classical OLAP operations (e.g., *slice, dice, roll-up, drill-down*) with the constraint that they should include at least one spatial condition from the spatial operation classes given above.

Spatial aggregation operators aggregate spatial (data) objects and returns a new composite spatial object. Some of the spatial aggregation operators are *union, convex hull, minimum bounding rectangle (MBR)* etc. These operators can be annotated at the schema level with QB4SOLAP (Fig. 9), and used on the instance level (applied on spatial measures) if the triple store supports spatial aggregation over spatial data types.

Topological relations are spatial Boolean predicates such as *intersects, within, contains, crosses* etc., that can be applied on two spatial objects, which return true or false as a result. Topological relations can be annotated at the schema level with QB4SOLAP (Fig. 9) between child and parent levels at a hierarchy step. Prior to annotating the topological relations at the schema level, topological relations should be derived between the child-parent level members at the instance level, by applying the set of topological relations from the QB4SOLAP definitions. Afterward, satisfied topological relations (the ones that return true) should be annotated at the schema level. Finally, numerical operations are spatial functions that are applied on spatial objects and returns a numeric value, e.g., *perimeter, area, distance*, etc.

Numerical operations can be applied directly on the instance level to spatial level attributes (with spatial data type), or to spatial measures (with spatial data type), and they are not annotated at the schema level. By using operations in this class we can reveal dynamic spatial hierarchy levels and new spatial members as we explain in the following example (Ex. 4) and shown in Fig.11.

City	**Customer**	**Supplier Sales**			**Total Sales**
		s1	s2	s3	
Düsseldorf	c1	8pcs.	–	3pcs.	11pcs.
	c2	10pcs.	–	–	10pcs.
Dortmund	c3	7pcs.	4pcs.	–	11pcs.
	c4	–	20pcs.	3pcs.	23pcs.
Münster	c5	–	–	30pcs.	30pcs.

Table 4: Sample (Instance) Data for Sales (adapted from [12, 15])

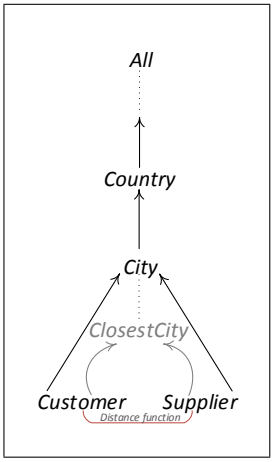


Fig. 11: Dyn. Hier. (adapted from [12, 15])

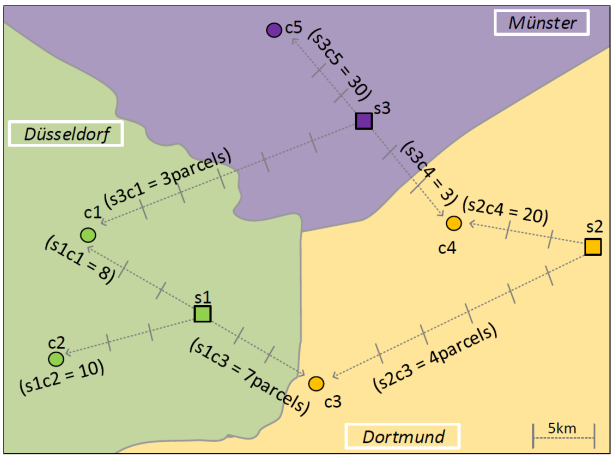


Fig. 12: Example Map of Sales (Instance) Data (adapted from [12, 15])

Example 4 (SOLAP example (s-roll-up) - Reproduced from [12, 15])
In this example we give an spatial extension to classical OLAP operator *roll-up*, which is used aggregate measures in order to get the data at a higher granularity level. Consider a *sales* fact cube with measures such as *number of sales, sales price,* etc. The fact cube has (spatial) dimensions for instance, *customer, supplier,* etc. with spatial hierarchy *geography*, which has *city, state, country, continent,* etc. as spatial levels. Using the classical roll-up operator, a user can *aggregate the total amount of sales to customers up to city level* (shown with straight arrows in Fig. 11).

Moreover, by using a spatial operation with roll-up we can query with s-roll-up, which can reveal new perspectives, such as, *total sales to customers by the city of the closest suppliers*. By using this s-roll-up operation, we create a *dynamic spatial hierarchy* during the query, with a spatial constraint to aggregate measures to the *closest city* of the suppliers (shown with curved arrows in Fig. 11). In order to demonstrate the difference between the classical roll-up and s-roll-up we give the instance data on the map in Fig. 12, with the number of sales made (in parcels) to each customer from the corresponding suppliers. The map also represents the distances with arrows between the customer and supplier locations in each city. Sale data instances between customers and suppliers are summarized in Table 4. We also give the calculated distances between customer and supplier locations in Table 5.

Finally, in Tables 6 and 7, we give the results of roll-up and s-roll-up operations, respectively. As shown in Table 5, customer *c3* is in city *Dortmund*, though its closest supplier is not in Dortmund but in *Düsseldorf*, which is

	Sup. City	Düsseldorf	Dortmund	Münster
Cust. City	**Sup.** / **Cust.**	s1	s2	s3
Düsseldorf	c1	15 km.s	45km.s	30 km.s
	c2	15 km.s	60 km.s	60 km.s
Dortmund	c3	*15 km.s*	30 km.s	45 km.s
	c4	45 km.s	15 km.s	*15 km.s*
Münster	c5	60 km.s	45 km.s	15 km.s

Table 5: Customer to Supplier Distance (km.s) (adapted from [12, 15])

supplier *s1*. Similarly, customer *c4* from Dortmund is closer to supplier *s3*, which is not from Dortmund city but from *Münster*. Dynamic spatial hierarchy changes the aggregation level city in s-roll-up, therefore we can have new analyses perspectives, which are normally not possible to reveal with traditional roll-up. Due to this new perspectives we observe different results in Tables 6 and 7.

City	Sales
Düsseldorf	21pcs.
Dortmund	34pcs.
Münster	30pcs.

Table 6: Roll-up (adapted from [12, 15])

City	Sales
Düsseldorf	25pcs.
Dortmund	20pcs.
Münster	33pcs.

Table 7: S-Roll-up (adapted from [12, 15])

SOLAP Semantics

We have described the formal SOLAP semantics of four common spatial OLAP operators: *s-slice*, *s-dice*, *s-roll-up*, and *s-drill-down* in details in Paper C. These formal semantics are necessary to be re-used in the SOLAP (to SPARQL) query generator algorithms. In order to give an overview of the SOLAP semantics, the following remark (Remark 1) gives the definition of traditional *dice* operator, definition (Def. 2) gives the definition of *s-dice* with high level SOLAP semantics, and finally Ex. 5 exemplifies the s-dice solap operator for different cases, which are all reproduced from [15].

Remark 1. (Dice - Reproduced from [15]) The traditional *dice* operator takes a cube and a Boolean condition ϕ, which returns a new cube containing only

the cells that satisfy the Boolean condition ϕ. Dice operation is analogous to relational algebra, *R selection*; $\sigma_{\phi}(R)$, but the argument is a cube not a relation. For example, the query "sales to customers of type LLC (Limited Liability Company)" is a dice operation. (Cube is the sales, dimension is the customer, and the Boolean condition is the customer type if they are LLC).

Definition 2. (S-Dice Reproduced from [15]) Similarly, the *s-dice* operator takes an n-dimensional cube $\mathcal{C}$ as an argument, which has the cube schema $\mathcal{CS} = (D, M, F)$ with the fact members $f \in FM$ as given in Def. 16 (Paper C). As a parameter s-dice takes a spatial Boolean predicate, which is denoted by $\phi^{\mathcal{S}}$. The s-dice operator keeps the cells of the cube $\mathcal{C}$ that satisfies the spatial predicate over spatial dimension levels l_s, attributes a_s, and measures m.

The semantics of the operator is defined as:
$\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}] = \mathcal{C}'$ where spatial predicate $\phi^{\mathcal{S}}$ can be applied on spatial level member values $\phi^{\mathcal{S}}(v_{l_s})$, spatial attribute values $\phi^{\mathcal{S}}(v_{a_s})$, measure values $\phi^{\mathcal{S}}(v_m)$ and/or a combination of these.

$\mathcal{SD}$ operator returns a sub cube $\mathcal{C}' \subseteq \mathcal{C}$, which has the schema $\mathcal{CS} = (D', M', F')$ where $D' = D$, $M' = M$, and $F' = F$. Unlike the s-slice operator, s-dice keeps all the dimensions D in the output cube $\mathcal{C}'$. The set of measures M and the fact type F also remains the same, though the new cube $\mathcal{C}'$ is a subset of the original cube $\mathcal{C}$ with filtered fact members $f \in FM'$, which is explained in the following.

The s-dice operator selects a subset FM' of the fact members' set $FM' \subseteq FM$ with respect to the spatial predicate $\phi^{\mathcal{S}}$ on level members as follows;

1. Spatial predicate on level values: $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge \phi^{\mathcal{S}}(v_{l_s})\}$.
2. Spatial predicate on level attribute values:
$FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) \wedge v_{l_s} \rightsquigarrow v_{a_s} : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge v_{l_s} \rightsquigarrow v_{a_s} \wedge \phi^{\mathcal{S}}(v_{a_s})\}$.
Note that the filtering the facts through level members can be done by v_{l_s} (level values) or attribute values v_{a_s} by applying the spatial predicate $\phi^{\mathcal{S}}$. Finally filtering of the facts is on associated measure values is defined in the following;
3. Spatial predicate on measure values of m_s: $FM' = \{f \in FM \mid \exists\, v_{m_s} \in Codomain(m_s) : f \rightsquigarrow v_{m_s} \wedge \phi^{\mathcal{S}}(v_{m_s})\}$.
For complex cases, i.e., combining these three types; the result set is also followed by combining the basic result sets.

Example 5 (S-Dice Example - Adapted from [15])
The s-dice operator can be implemented on the level and attribute values by filtering level members in the cube or on measures by filtering the facts in the cube. In both cases, the spatial predicate $\phi^{\mathcal{S}}$ is used.

The query for the s-dice operator could be "sales to customers, which are located within 5 km distance from their city center" where the s-dice is on level members by filtering the customer level. The spatial predicate $\phi^{\mathcal{S}}$ can be interpreted in two different ways (See Table 8 for comparison of their query run times in conclusion and discussion section (Sect. 3.5)).

S-Dice (1) The first method is assuming a buffer area of 5 km from the coordinates of the city center and checking customers' locations by *within* operator from topological relations $\phi^{\mathcal{S}} \in \mathcal{T}_{rel}$ if it meets the condition. The following SPARQL query shows the implementation of this method on level members.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city ;
      gnw:customerGeo ?custGeo .
   ?city gnw:cityGeo ?cityCentGeo .
FILTER (bif:st_within (?custGeo, ?cityCentGeo,
5))}
```

S-Dice (2) Second method is checking if the distance from a customer location to the corresponding city center is less than 5 km, by using *distance* function from numeric operations $\mathtt{f}^{\mathcal{S}} \in \mathcal{N}_{op}$. In this case the spatial predicate $\phi^{\mathcal{S}}$ is a combination of a spatial function $\mathtt{f}^{\mathcal{S}}$ and a regular Boolean predicate ϕ. Spatial function is *distance* from numeric operations and the predicate is *less than* ($<$). The following SPARQL query shows the implementation of this method for s-dice on level members.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city ;
      gnw:customerGeo ?custGeo .
   ?city gnw:cityGeo ?cityCentGeo .
BIND (bif:st_distance (?custGeo, ?cityCentGeo)
AS ?distance) FILTER ( ?distance < 5 ) }
```

3.4 SOLAP Query Generation

In the previous section we have mentioned about two SOLAP operators: s-roll-up in Ex. 4 to give a general understanding of SOLAP operators and s-dice in Ex. 5 to clarify the given semantics of spatially extended dice operator with Remark 1 and Def. 2. In order to keep it simple, we have chosen only one SOLAP query generation algorithm to be explained in the summary and that is *s-roll-up*[21].

The SPARQL queries that are generated from SOLAP operators are denoted with Q and has the form "Q = `SELECT` R `WHERE` GP", where GP represents a graph pattern that contains triple patterns and R is the set of parameters, which are returned as a result of the query. As can be observed from the above s-dice examples (Ex. 5) `skos:broader` property is used to define the roll-up relation between levels (e.g., {`?cust skos:broader ?city`}). In order to represent roll-up paths for hierarchy levels of a dimension, we defined a helper function *RUPath* (Algorithm 1), which creates these roll-up paths included in GP in the body of the `WHERE` clause.

Algorithm 1: $RUPath(\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?a_s, ?f) : GP$ - Adapted from [15]

Input: $\mathcal{G}^S_{(\mathcal{C})}$, l_b, l_s, a_{ID},?a_s, ?f
Output: GP

1 **begin**
2 GP = (?f `rdf:type qb:Observation`)
3 $GP = GP \cup$ (?f $id^S(a_{ID})$?l_b $\wedge$?l_b `qb4o:memberOf` $id^S(l_b)$)
4 **foreach** $(id^S(l_c), id^S(l_p)) \in \mathcal{G}^S_{(\mathcal{C})} \mid l_p \sqsubseteq l_s$ **do**
5 $GP = GP \cup$ (?l_b `skos:broader` ?l_p $\wedge$?l_p `skos:broader` ?l_s)
6 let $GP = GP \cup$ (?l_s $id^S(a_s)$?a_s)
7 **return** GP

Roll-up Path. The roll-up path triple pattern is created as a path-shaped join of triples with the following form: $\{(s_1\ p_1\ o_1), (o_1\ p_2\ o_2), (o_2\ p_2\ o_3), \ldots (o_{n-1}\ p_2\ o_n)\}$. The root of the pattern is s_1, which corresponds to fact members (a.k.a. observations) (Line 2 in Algorithm 1). The predicate p_1 is $id^S(a_{ID})$, which is the identifier of attribute IDs and that links the facts with identifiers of the base level members $id^S(l_b)$ (Line 3 in Algorithm 1). The first level member variable (base level member - ?l_b) corresponds to o_1, which rolls-up to its parent level member o_2 with `skos:broader` predicate $p2$. This

[21] All of the query generation algorithms for four main SOLAP operators can be found in detail in Paper C.

Algorithm 2: *SRUGenerator*($\mathcal{G}^{I}_{(\mathcal{C})}$,$\mathtt{f}^{\mathcal{S}}(L(d_s))$, $agg(m)$) : Q - Adapted from [15]

Input: $\mathcal{G}^{I}_{(\mathcal{C})}, \mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)$
Output: Q
1 **begin**
2 $Q = \emptyset$; GP = `RUPath`($\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?$\mathrm{a}_s$,?f)
3 $GP = GP \cup$ (?f $id^{\mathcal{S}}(m)$?m)
4 **for** $\mathtt{f}^{\mathcal{S}}(L(d_s))$ **do**
5 $GP' = $ `RUPath`($\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?$\mathrm{a}_s$,?f);
$Q' = \emptyset$
6 $GP' = GP' \cup$ (`BIND` $\mathtt{f}^{\mathcal{S}}(x)$ `AS` ?v_x)
7 Q' = `SELECT` ?x (`AGG`(?v_x) `AS` ?v_y) `WHERE` GP' `GROUP BY` ?x
8 $GP = GP \cup Q' \cup$ (`FILTER` ?x = ?a_s && $\mathtt{f}^{\mathcal{S}}(x)$ = ?v_y)
9 let $l_s = l'_s$
10 **return** Q = `SELECT` ?f ?l'_s `AGG`(?m) `WHERE` GP `GROUP BY` ?f ?l'_s

continues for each child-parent level member (Line 4 in Algorithm 1) until the target level is reached l_s, which is the last variable (?l_s in Line 5) in the path and corresponds to o_n in the triple pattern. Target level l_s defines the granularity of the results to be returned from the (spatial) OLAP operation. Finally, the graph pattern GP is added and returned with the required spatial attribute parameters in Line 6[22]. The helper function *RUPath* is used in all of the SOLAP generations algorithms to create the graph pattern retuned by the SPARQL query.

S-Roll-up Generator. A high-level SOLAP expression is defined in Paper C (Def. 19) with SOLAP semantics as: $\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)]$. The parameter $\mathtt{f}^{\mathcal{S}}(L(d_i))$ represents a spatial function on spatial level members of a dimension level ($L(d_i)$) and $agg(m)$ represents an aggregate function on measures. In order to illustrate the algorithm steps, we use the s-roll-up example given in Ex. 4: "Total amount of sales to customers by city of the closest suppliers". The following text is adapted from [15], which sketches main steps *line by line* from Algorithm 2.

Lines 2, 3 (Adapted from [15]). Build the roll-up path using helper function `RUPath`. In addition to the variables given in the `RUPath` function, we

[22]In order to represent varying (changeable) parameters of the triple pattern at the instance level such as fact/level members and parameter values given by the user from the other parameters in the algorithm, we represent those with questions marks before the variable name, e.g., ?f, ?l_x, ?a_s, etc.

also need to consider measures and measure value variables (Line 3) since we aggregate the measures. A measure is specified in the following listing of the running example as `gnw:salesAmount`. The following lines are added to the graph pattern GP:

```
{?obs rdf:type qb:Observation ;
    gnw:customerID ?cust ;
    gnw:supplierID ?sup ;
    gnw:salesAmount ?sales .
?cust qb4o:memberOf gnw:customer ;
    gnw:customerGeo ?custGeo ;
    skos:broader ?city .
?sup qb4o:memberOf gnw:supplier ;
    gnw:supplierGeo ?supGeo ;
    skos:broader ?city .
?city qb4o:memberOf gnw:city .
```

Line 4 (Adapted from [15]). Build inner select subquery to apply the spatial function $\mathtt{f}^{\mathcal{S}}$ on the spatial level members $L(d_i)$ (i.e., Customer, Supplier). In the example, we will use this information to create a dynamic spatial hierarchy from the Customer to the City level.

Line 5 (Adapted from [15]). Call `RUPath` for the inner select subquery to link the geometry attributes of base level members with different variables and create a graph pattern GP' for the inner select. The following lines are added to the graph pattern GP':

```
{?obs rdf:type qb:Observation ;
    gnw:customerID ?cust1 ;
    gnw:supplierID ?sup1 .
?sup1 gnw:supplierGeo ?sup1Geo .
?cust1 gnw:customerGeo ?cust1Geo .
```

Line 6 (Adapted from [15]). Build the bind statement in order to calculate the spatial function $\mathtt{f}^{\mathcal{S}}(L(d_s))$ on spatial level members. For the running example the spatial function is `st_distance`. The following lines are added to the graph pattern GP':

```
BIND (bif:st_distance(?cust1Geo, ?sup1Geo)
AS ?distance)}
```

Line 7 (Adapted from [15]). Generate the inner select query Q' using graph pattern GP' (Lines 5 and 6). Select the corresponding level members (Customer level for the running example) and group them in a group by statement on the selected level members. Note that this is where

the spatial function $\mathtt{f}^{\mathcal{S}}(L(d_i))$ is called with a wrapper expression (e.g., MIN, MAX, etc.) to find the closest distance. The following lines illustrate the inner select query Q':

```
Q‘ = {SELECT ?cust1 (MIN(?distance) AS
?minDistance) WHERE GP‘
   GROUP BY ?cust1}
```

Lines 8, 9 (Adapted from [15]). Build the filter statement for the whole query based on the output of the spatial function, which is calculated in the inner select subquery. Then, add the filter and inner select subquery to the main graph pattern GP' (Line 8). The filter statement for the running example is :

```
FILTER (?cust = ?cust1 && bif:st_distance
(?custGeo, ?supGeo) = ?minDistance)}
```

Note that in Line 9, the spatial target level l_s (City) is altered to a dynamic spatial level l'_s since applying the spatial function creates a dynamic hierarchy.

Line 10 (Adapted from [15]). Generate query Q for computing the facts $f \in FM'$ based on graph pattern GP created in the previous steps. The measures are also aggregated at the spatial target level (closest City, which is dynamically selected). The group by statement is applied to the fact members and target level members. In our running example, we obtain the following case for the generated s-roll-up query Q.

Finally, the complete generated SPARQL query from the above steps is given with the following listing in Ex. 6.

Example 6 (S-Roll-up in SPARQL - Adapted from [15])
Graph pattern GP' (for the subquery in inner select) is created in Lines 15-22, and the graph pattern GP (for the whole query) is created in Lines 3-24.

```
1  Q = SELECT ?obs ?city (SUM(?sales) AS
2  ?totalSales) WHERE
3  { ?obs rdf:type qb:Observation ;
4     gnw:customerID ?cust ;
5     gnw:supplierID ?sup ;
6     gnw:salesAmount ?sales .
7  ?cust qb4o:memberOf gnw:customer ;
8     gnw:customerGeo ?custGeo ;
9     gnw:customerName ?custName ;
10    skos:broader ?city .
```

```
11 ?city qb4o:memberOf gnw:city .
12 ?sup gnw:supplierGeo ?supGeo .
   # Inner Select for the distance function
13    { SELECT ?cust1 (MIN(?distance) AS
14    ?minDistance) WHERE
15    { ?obs rdf:type qb:Observation ;
16       gnw:customerID ?cust1 ;
17       gnw:supplierID ?sup1 .
18    ?sup1 gnw:supplierGeo ?sup1Geo .
19    ?cust1 gnw:customerGeo ?cust1Geo .
20    BIND (bif:st_distance( ?cust1Geo, ?sup1Geo )
21    AS ?distance)}
22    GROUP BY ?cust1 }
23    FILTER (?cust = ?cust1 && bif:st_distance
24    (?custGeo, ?supGeo) = ?minDistance)}
25 GROUP BY ?city ?obs
```

Nested SOLAP. It is very common among data warehouse users to query data warehouses with nested (spatial) OLAP queries. Generally, a nested set of SOLAP operators can be created as an expression with an s-dice on top of several s-roll-ups that is on top of one or more s-slices on top of an s-dice, e.g. [15], *(s-dice*$_2$*(s-roll-up*$_1$*(...s-roll-up*$_k$*(s-slice*$_1$*(... s-slice*$_n$*(s-dice*$_1$*(C)))))))*.

In order to perform nested SOLAP operations in SPARQL by using the SOLAP generation algorithms, we have defined a set of principles to be considered in Algorithm 3. These principles are simply necessary for two reasons: 1) in order to follow the general order the SOLAP operators in a nested SOLAP expression as explained in the previous paragraph and 2) in order to address the syntactic details in SPARQL queries and graph patterns that are generated from the SOLAP generator algorithms, i.e., ordering of SPARQL clauses and functions (`SELECT`, `FILTER`, `BIND`, `GROUP BY`, etc.). The following principles are adapted from [15].

Principle 1: Perform s-dice in the beginning or at the end.

Principle 2: If there are several s-roll-up or s-slice operations call their generator algorithms repeatedly.

Principle 3: Always separate `FILTER` clauses when a SOLAP generator algorithm is used. Enumerate separated `FILTER` clauses. If a SOLAP operator is the final function added to the graph pattern, do not separate the `FILTER` clause.

Principle 4: Build the final graph pattern with the separated and enumerated `FILTER` clauses with respect to Principle 3.

Principle 5: Drop the main `SELECT` clause from each SOLAP generator algorithms and build only one `SELECT` that is added to the query at the end.

Principle 6: Separate the `GROUP BY` clause and `AGG` functions from the s-roll-up generator algorithms (and enumerate them), and build add them to the main (outer) `SELECT` clause at the end.

As a starting point to write a nested SOLAP query we have formulated a simple nested form, which is the most typical pattern: *(s-roll-up (s-slice (s-dice(C))))*. By following the above principles, *WriteSPARQL* algorithm is given as pseudo-code that takes the high-level SOLAP operator definitions as input (Algorithm 3).

In Algorithm 3, Lines 4 and 6, separating and enumerating `FILTER` clauses can be observed with respect to *Principle 3*. Accordingly with *Princliple 4*, the final graph pattern is built by the union of separated and enumerated `FILTER` clauses in Line 8. In Lines 4,6, and 9 `SELECT` statements are dropped from the SOLAP operator algorithms (*S-DiceGenerator, S-SliceGenerator,* and *SRU-Generator*) with respect to *Principle 5*, and a single`SELECT` is built at the end added to the query in Line 10. Similarly, `GROUP BY` clause and `AGG` functions are dropped from *SRUGenerator* algorithm in Line 9 and added built in the outer `SELECT` with the main query in Line 10 as suggested in *Principle 6*.

Algorithm 3: *WriteSPARQL*$((\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)](\mathcal{SS}(\mathcal{C})[l_b, l_s, v_{a_s}](\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}])))) : Q$ - Adapted from [15]

Input: $(\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_s)), agg(m)](\mathcal{SS}(\mathcal{C})[l_b, l_s, v_{a_s}](\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}])))$
Output: Q

1 **begin**
2 $Q = \emptyset$; GP = `RUPath`$(\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?a$_s$,?f)
3 $GP = GP \cup$ (?f $id^{\mathcal{S}}(m)$?m)
4 GP^1 = `S-DiceGenerator`$(\mathcal{G}^{I}_{(\mathcal{C})}, \phi^{\mathcal{S}})$ \ `FILTER`^{1} \ `SELECT`
5 $GP = GP \cup GP^1$
6 GP^2 = `S-SliceGenerator`$(\mathcal{G}^{I}_{(\mathcal{C})}, v_s, l_b, l_s)$ \ `FILTER`^{2} \ `SELECT`
7 $GP = GP \cup GP^2$
8 $GP = GP \cup$ `FILTER`$^1 \cup$ `FILTER`$^2 \cup$
9 `SRUGenerator`$(\mathcal{G}^{I}_{(\mathcal{C})}, \mathtt{f}^{\mathcal{S}}(L(d_s)), agg(m))$ \ `SELECT` \ `GROUP BY`^{1} \ `AGG`^{1}
10 **return** Q = `SELECT` ?l$'_s$ `AGG`^{1}(?m) `WHERE` GP `GROUP BY`^{1} ?l$'_s$

Remark that in the algorithm, the main graph pattern GP is initiated with

the *RUPath* helper function in Line 2, and incremented with a triple pattern for selected measures coming from the *SRUGenerator* algorithm in Line 3. After dropping the `FILTER`, `SELECT`, etc. clauses from the SOLAP generator algorithms in each step, sub-graph patterns are created and enumerated (GP^1, GP^2), which are incrementally added to the main graph pattern GP in Lines 5 and 7. In the following an example is given from Paper C, showing the output of *WriteSPARQL* algorithm for generating a nested SOLAP query [15]. We use the running use examples for *s-roll-up* and *s-dice* given earlier (Examples 4, 5, 2).

Example 7 (($(^3$*s-roll-up* $(^2$*s-slice* $(^1$*s-dice(C)*$)))$) - Adapted from [15])
[1]Get the subcube graph of customer that are located within a 5 km distance from their city center, [2]slice on the customers of the largest country, (which drops the dimension and leave out all the other countries) and [3]get the total amount of sales of customers by the city of their closest suppliers (aggregates the measure Sales amount from Customer to Closest City level). The query is written starting from the innermost operator s-dice to the outermost operator s-roll-up.

```
1  SELECT ?city (SUM(?sales) AS ?totalSales)
2  WHERE {
3     ?obs rdf:type qb:Observation ;
4        gnw:customerID ?cust ;
5        gnw:supplierID ?sup ;
6        gnw:salesAmount ?sales .
7     ?cust qb4o:memberOf gnw:customer ;
8        gnw:customerGeo ?custGeo ;
9        skos:broader ?city .
10    ?sup qb4o:memberOf gnw:supplier ;
11       gnw:supplierGeo ?supGeo ;
12       skos:broader ?city .
13    ?city qb4o:memberOf gnw:city ;
14       gnw:cityGeo ?cityCentGeo ;
15       skos:broader ?country .
16    ?country qb4o:memberOf gnw:country ;
17       gnw:countryGeo ?countGeo .
18    ?city gnw:cityGeo ?cityGeo .
   ### 1.Inner select for (S-SLICE)
   ### Find the largest country
19    {SELECT ?x (MAX(?area) as ?maxArea)
20    WHERE {
21       ?obs rdf:type qb:Observation ;
22          gnw:customerID ?cust .
23       ?cust qb4o:memberOf gnw:customer ;
24          skos:broader ?city .
```

```
25        ?city skos:broader ?country .
26        ?country gnw:countryGeo ?x .
27     BIND(bif:st_area(?x) as ?area)}}
   ### 2.Inner select for (S-ROLL-UP)
   ### Find the closest suppliers to customers
28     { SELECT ?cust1 (MIN(?distance) AS
29     ?minDistance) WHERE {
30        ?obs rdf:type qb:Observation ;
31           gnw:customerID ?cust1 ;
32           gnw:supplierID ?sup1 .
33        ?sup1 gnw:supplierGeo ?sup1Geo .
34        ?cust1 gnw:customerGeo ?cust1Geo .
35     BIND (bif:st_distance(?cust1Geo, ?sup1Geo)
       AS ?distance)}
36     GROUP BY ?cust1 }
   ### FILTER for S-DICE, to get a subcube
37 FILTER (bif:st_within (?custGeo, ?cityCentGeo, 5))
   ### FILTER for S-SLICE, the 1st inner SELECT
38 FILTER (?countGeo = ?x)
   ### FILTER for S-ROLL-UP, the 2nd inner SELECT
39 FILTER (?cust = ?cust1 && bif:st_distance
   (?custGeo, ?supGeo) = ?minDistance)}
40 GROUP BY ?city
```

3.5 Conclusion and Discussion

In order to query with SOLAP operators, a well-defined spatial multi-dimensional data model is required. We have introduced QB4SOLAP Vocabulary - a spatial extension to the QB4OLAP multi-dimensional RDF Vocabulary. We have explained the highlights of the MD concepts of QB4OLAP and the spatial extensions introduced with QB4SOLAP. We have briefly mentioned our journey of building up the QB4SOLAP vocabulary developed from the earlier two versions. Describing the spatial extensions of a data warehouse model on the Semantic Web requires good knowledge on MD concepts and their availability in prominent RDF vocabularies (i.e., RDF Data Cube Vocabulary and QB4OLAP), and a good understanding of spatial functions, topological relation models[23] and their standards for the Semantic Web (i.e., GeoSPARQL as an OGC standard). We had to make design decisions for spatial extensions of QB4SOLAP and definition of SOLAP operations, which are based on a classification of spatial operations in three groups: spatial aggregation, topological

[23]RCC8 - Region Connection Calculus [27] and DE-9DIM - Dimensionally Extended Nine-Intersection [9] models describe possible Boolean relations of two geometries, in Euclidean space and two-dimensional space, respectively.

relation, and numeric operation.

After introducing our definition of SOLAP, we explained the possible new analyses perspectives with a running use case example. For writing similar SOLAP queries in SPARQL, the use case data should be annotated with QB4SOLAP both at the schema level and the instance level, since OLAP query structure uses explicit schema concepts of MD models (e.g., aggregate/roll-up measures to a higher level along a hierarchy, slice/remove a dimension from the fact cube, etc.). QB4SOLAP vocabulary allows users to annotate and publish spatial multi-dimensional RDF data. Thus the users can query spatial RDF endpoints with SOLAP operators in SPARQL.

Due to the complexity of the SPARQL query language for inexperienced users, high-level SOLAP operators are required to be parsed and generated in SPARQL. In order to achieve that, we have formalized the RDF semantics of multi-dimensional cube concepts of QB4SOLAP schema and instance data which is exemplified with a running use case. As an example, we have shown the hierarchy steps QB4SOLAP terms in Ex. 1. We have also defined high-level SOLAP semantics of four common SOLAP operators (s-slice, s-dice, s-roll-up, and s-drill-down), which are given together with SPARQL example queries from the running use case. An example of a SOLAP operator is given with Def. 2 and Ex. 5 for s-dice operator. Finally, by re-using the semantics of SOLAP operators and multi-dimensional RDF data cubes definitions in QB4SOLAP, we have provided algorithms for generating spatially extended SPARQL algorithms from single or nested SOLAP operators. The query runtimes (in seconds) tested against an instance of the running use case data set (GeoNorthwind) is given in Table 8 below.

Table 8: Runtimes in seconds (Reproduced from [15])

Examples	**SOLAP Operators**	**Query Runtime**
Ex. 12 - Paper C	*(s-slice($\mathcal{C}$))*	0.07
Ex. 5 (1), Ex. 13.1 - Paper C	*(s-dice($\mathcal{C}$))*	0.09
Ex. 5 (2), Ex. 13.2	*(s-dice($\mathcal{C}$))*	1.01
Ex. 6, Ex. 14 - Paper C	*(s-roll-up($\mathcal{C}$))*	2.03
Ex. 15 - Paper C	*(s-drill-down($\mathcal{C}$))*	1.86
Ex. 7, Ex. 16 - Paper C	*(s-roll-up (s-slice (s-dice($\mathcal{C}$))))*	3.04

GeoNorthwind dataset (conceptual schema is given in Fig. 10) is annotated with QB4SOLAP and in total *48677* triples are produced and published to an RDF endpoint[24] [15]. Each SOLAP operator in Table 8 takes the defined QB4SOLAP cube as a parameter $\mathcal{C}$. This corresponds to the published

[24]**RDF Endpoint:** `http://lod.cs.aau.dk:8890/sparql`, **Software set-up:** Virtuoso Open Source Edition (Column Store and multi threaded) Version 7.2.5 running on an Ubuntu 14.04 server with 2.30GHz CPU and 16 GB RAM **SOLAP Queries in SPARQL:** `http://extbi.cs.aau.dk/SOLAP4SW/queries`

GeoNorhtwind RDF triples. As can be seen from the table above, s-slice took 0,07 seconds to execute, which is quite an efficient SOLAP operator since it contains a `FILTER` statement at a specified spatial level. The queries given in Ex. 5 for s-dice operator differs as 0.09 sec. for (1) and 1.01 for (2). The first query of s-dice in SPARQL is more efficient than the second one, even though the results are the same. The first s-dice is performed by filtering the geometry instances (of customers) from a specified given point (city center geometry), which are within a 5 km. buffer area. In this way of implementing s-dice SOLAP operator, a topological relation $\mathcal{T}_{rel}$ (`st_within`) is used. The second s-dice is performed by calculating every customer instance distance to the city center with a spatial numerical operation $\mathcal{N}_{op}$ (`st_distance`), and then a filter applied to retrieve those, which has less than 5 km distance to the city center. The second approach of s-dice is more expensive since we have to measure all the distances between each customer and the city center. The s-roll-up example given in Ex. 6 has a runtime of 2.03 seconds. The reason why it has a longer response time than s-dice and s-slice operators is due to its complexity, where both spatial functions (e.g. `st_distance` to calculate the distance between customer and suppliers) and aggregate operators (e.g., `SUM` of sales) are used in the query. Next operator, s-drill-down works in a very similar way as s-roll-up and have a close runtime measurement as 1.86 seconds. The nested SOLAP query given in Ex. 7 has the longest run time measured as 3.04, due to nesting of several SOLAP operators in the query. Naturally, the number of triple patterns is much more than a single SOLAP operator in a nested query, thus, the query run time has the highest measurement.

Our future vision of SOLAP and its tools on the Semantic Web is depicted in Fig. 13. In conclusion, we built the conceptual and the architectural groundwork for Geo-semantic (Spatial RDF) Data Warehouses by proposing QB4SOLAP vocabulary, and demonstrating examples and high-level SOLAP semantics on an applied spatial use case. We have also shown how the high-level SOLAP operators can be translated into SPARQL by the SOLAP generator algorithms given in pseudo-code (Paper C).

Even though the QB4SOLAP vocabulary, defined SOLAP semantics, and SPARQL query generator algorithms are important contributions towards Geo-semantic Data Warehouses, they are not enough itself, since annotating and querying spatial data on the Semantic Web is limited to experts with SPARQL and RDF knowledge. Tools such as GeoSemOLAP that can generate SPARQL queries from high-level SOLAP expressions interactively via GUI are required to allow users to perform SOLAP on the SW without knowledge of SPARQL. Another tool that is required is RDF2SOLAP to automate the QB4SOLAP annotation from the existing RDF endpoints.

Thus, in the upcoming stages within the scope of this project, we develop and offer solutions for the practical and technical implementation of Geo-

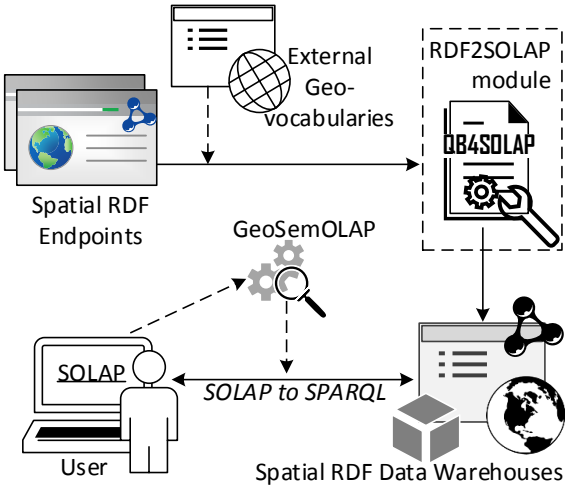


Fig. 13: Our future vision of SOLAP on the SW - Adapted from [15]

semantic Data Warehouses and SOLAP operations on the Semantic Web by developing these tools.

4 The GeoSemOLAP Framework

This section gives an overview of (demo paper) Paper D [13], where we demonstrate the implementation and workflow of SOLAP generator algorithms from Section 3.

4.1 Motivation and Problem Statement

The increasing popularity of publishing Linked Open Data (LOD) on the SW from the public sector, made data sets from spatial and governmental domains available in RDF format. In order to enable querying such data sets with spatial and analytical DW queries (a.k.a. SOLAP), users and decision-makers need to have good knowledge of SPARQL query language and syntax along with SOLAP semantics. We have shown in Section 3.3, translating SOLAP operator semantics to SPARQL queries requires good knowledge in MD concepts and RDF parsing. Moreover, generated SOLAP operators in SPARQL query syntax are not intuitive and easily readable. Potential decision-makers and users are not often fully familiar with SPARQL syntax to perform SOLAP operators (in SPARQL) through a spatial MD RDF data endpoint. Mostly the users are well-aware about the MD concepts of the data

model and know how to query DWs with high-level spatial OLAP operators, which are similar in principle to Multidimensional Data Expressions (MDX) syntax[25].

In order to achieve to our vision of tool oriented future for SOLAP on the SW (depicted in Fig. 13), we have to develop and provide tools (such as *GeoSemOLAP*) for end-users to lower the entry barrier for querying spatial RDF endpoints with SOLAP operators.

In the following, we recap the SOLAP example (s-roll-up), which is very similar to Ex. 4 except the city names. In addition, to show the complexity of SOLAP on the SW, we present the s-roll-up query in SPARQL, highlighted with facts and level members. The highlighted fact and level members from the SPARQL query can be identified as MD concepts of a high-level SOLAP expression for s-roll-up.

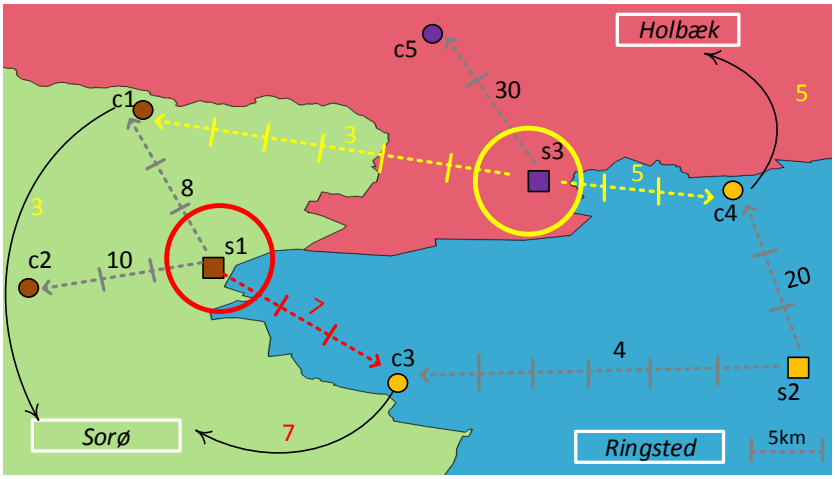


Fig. 14: Example map of sales data (Reproduced from [13])

Example 8 (SOLAP example (s-roll-up) in SPARQL - Adapted from [13]) The example scenario is very similar to the previously given SOLAP example (Ex. 4). Fig. 14 shows a map with amount of sales between customers *(c1, c2, . . . , c5)* and suppliers *(s1, s2, s3)* in three Danish cities *(Sorø, Holbæk, Ringsted)*. In the background story of this scenario, an export company records its sales data[26] in a spatial data warehouse. The company decides to publish the spatial data warehouse on the Semantic Web for further

[25]MDX is an industry-standard query and calculation language used to retrieve data from OLAP databases [23].

analysis possibilities and providing transparent access to all of its branches and customers.

An end-user/analyst would like to obtain: “The total sales to customers grouped by cities of their closest supplier". The dataset contains the corresponding information for the event of each sale about the involved customer and supplier, their city, location, and the number of sold goods. However, the dataset does not contain information about the distances between customers and suppliers, hence, we have to find the distances between customer and supplier points using a spatial function (i.e., distance) during query runtime. Based on the acquired information from the spatial function (closest distances between customer-supplier points) we can aggregate the total sales. Technically, this corresponds to the SOLAP operator: s-roll-up.

As can be seen from the following SPARQL query, performing this s-roll-up operation on the SW, requires good knowledge of understanding the SPARQL query syntax involving several triple patterns with many variables. Moreover, including and handling the spatial functions makes the SPARQL query even more complicated, which can be easily overwhelming for inexperienced users (on the SW).

```
SELECT ?obs ?supCity (SUM(?sales) AS ?totalSales)
WHERE {?obs rdf:type qb:Observation ;
   gnw:customerID ?cust ;
   gnw:supplierID ?sup ;
   gnw:salesAmount ?sales .
?cust qb4o:memberOf gnw:customer ;
   gnw:customerGeo ?custGeo .
?sup qb4o:memberOf gnw:supplier;
   gnw:supplierGeo ?supGeo ;
   skos:broader ?supCity .
?supCity qb4o:memberOf gnw:city .
# Inner select for the distance function
  {SELECT ?cust1 (MIN(?distance) AS ?minDistance)
  WHERE {?obs rdf:type qb:Observation ;
   gnw:customerID ?cust1 ;
   gnw:supplierID ?sup1 .
   ?sup1 gnw:supplierGeo ?sup1Geo .
   ?cust1 gnw:customerGeo ?cust1Geo .
   BIND (bif:st_distance(?cust1Geo, ?sup1Geo)
   AS ?distance)}
   GROUP BY ?cust1 }
   FILTER (?cust = ?cust1 && bif:st_distance
   (?custGeo, ?supGeo) = ?minDistance)}
GROUP BY ?supCity ?obs
```

Motivated by the need for an easy to use tool (for non-SW experts), in order to query spatial semantic data warehouses via SOLAP operators, we have developed *GeoSemOLAP* framework.

4.2 Understanding Spatial Semantic Data Warehouse Queries

GeoSemOLAP requires the semantic information about the schema of the spatial multi-dimensional data set in order to generate the SPARQL queries in an automated way. Therefore, GeoSemOLAP is developed using QB4SOLAP semantics (Section 3.3). QB4SOLAP[27] describes both multi-dimensional concepts and spatial concepts, which is built on top of existing RDF Data Cube (QB)[28] and QB4OLAP[29] vocabularies.

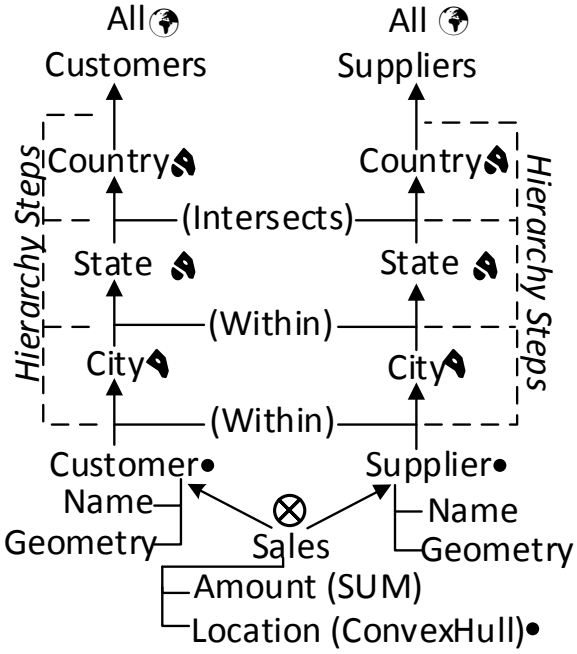


Fig. 15: Northwind spatial data cube members (symbols next to level names represent spatial characteristics of level members, e.g., point, polygon, and multi-polygon.) - (Reproduced from [13])

In Fig. 15, a sample schema of the running use case data is illustrated with

[27] **QB4SOLAP:** `https://w3id.org/qb4solap#`
[28] **RDF Data Cube:** `https://w3.org/TR/vocab-data-cube/`
[29] **QB4OLAP:** `https://lorenae.github.io/qb4olap/`

spatial concepts and MD concepts focused on Ex. 8. This schema is derived from the example given earlier in Fig. 10. Sales are the center of analysis which are called *observation (facts)* and have some associated *measures* such as Amount and Location. These measures are defined with corresponding aggregate functions; `SUM` for Amount and `ConvexHull` for Location. Location is a *spatial measure*, therefore the corresponding aggregate function is defined as `ConvexHull`, where location points can be aggregated as a convex-shaped envelope. These functions are used for aggregating measure values during roll-up operations to an upper *level* in the *dimension*, e.g., City to State level. (Fig. 15 captures only the spatial dimensions (Customer and Supplier) from the full conceptual schema of Northwind spatial data warehouse that was given in Fig. 10). Each roll-up relation between levels are defined as *hierarchy steps*, where *spatial* hierarchy steps are defined with a topological relation between its levels, e.g., City level is `WITHIN` State level, State level `INTERSECTS` Country level, etc.

The high-level SOLAP representation of the running query (Ex. 8) "Total sales to customer grouped by the city of their closest supplier" is S-ROLL-UP (Sales, [DISTANCE(Customer, Supplier)] $\rightarrow$ ClosestCity, SUM(SalesAmount)). This way of formulating an s-roll-up query is very common and more intuitive among decision-makers, without tackling with the SPARQL triple patterns, variables, and a complex syntax. The first line in the given query example (Ex. 8) specifies the variables to be returned as output from the outer `SELECT`: sales observations - `?obs` and supplier city - `?supCity`, and total amount of sales given with the aggregate function `SUM` on measure `?sales`.

The triple patterns between Lines 2-11 show a roll-up path as described with Algorithm 1 as a path-shaped join of triples. The roll-up path links the initial triple pattern, which is the center of analysis - sales observations (`?obs`) in Line 2 to target levels customer and supplier (`?cust`, `?sup`) in Lines 3 and 4. These levels are the base levels of the previously mentioned spatial dimensions: customer and supplier. The sales amount measure is also linked to the graph pattern[30]. The measure - sales amount is also linked to the graph pattern (`?sales`) in Line 5. In order to find the closest supplier cities to the customer, we will need the geometry attributes of spatial levels city and supplier. Thus, we need to acquire them as well in the graph pattern (Lines 7, and 9). Lines 6, 8, and 11 mediates spatial levels of the spatial hierarchy (Fig. 15). The supplier city is the target level to roll-up (Line 10). In order to find the closest supplier cities, the remaining of the query with the inner `SELECT` calculates the distances between customer and supplier geometries (Lines 18 and 19) to find the suppliers with the closest distance to customers (Lines 21 and 22).

[30] **Remark:** a set of RDF triple(s) and triple patterns are called RDF graph and graph pattern (Def. 4).

4.3 System Architecture and GeoSemOLAP Workflow

The system architecture of GeoSemOLAP is depicted in Fig. 16. The frontend architectural component of GemSemOLAP is *Graphical User Interface (GUI)*, which is for end-users to interact with the tool. *Metadata Manager* uses QB4SOLAP vocabulary and the use case data set schema in order to generate queries from high-level SOLAP concepts (spatial MD query elements) entered by the end-user via GUI. *Query Generator* works in a well-integrated manner with the metadata manager. The last two architectural components of GeoSemOLAP at the back-end are *Data Processor* and *SPARQL Endpoint*. GeoSemOLAP is developed with Javascript, HTML and CSS. To visualize the spatial data and maps, Leaflet API[31], and to store and query the RDF data from an endpoint Virtuoso Triple Store (Open Source Edition 7.2) is used.

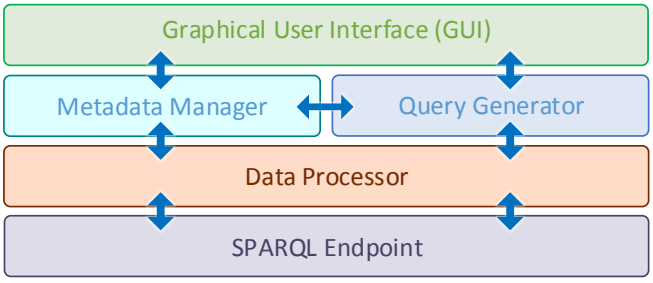


Fig. 16: GeoSemOLAP architecture (Reproduced from [13])

The following figure illustrates the workflow for users on how they can interact with GemSemOLAP (Fig. 17). Initially, the user selects a SOLAP operator. Based on the selected SOLAP operator, several options for spatial and MD elements (e.g., spatial levels, attributes, spatial functions such as distance, within) are displayed in the drop-down menus in order to complete the operator. Once the SOLAP operator is completed, these steps can be repeated for writing a nested SOLAP query. Hence, some SOLAP operators have the option to input coordinates from a map (such as s-slice in Fig. 18b), GeoSemOLAP displays a snippet of a map when the operator is selected.

Then, the user can click on the desired location on the map to indicate the coordinates that the query should take as an input. The third step is entirely operated by GeoSemOLAP automatically when the user finalizes selecting the SOLAP query elements, where a SPARQL query is generated from the formulated SOLAP operator(s) by the user. If the user has familiarity with the SPARQL syntax, she can easily edit the query and parameters from the generated query template (Fig. 18c) in the fourth step. When the user press *Run Query* button the generated query is sent the SPARQL endpoint and ex-

[31] http://leafletjs.com

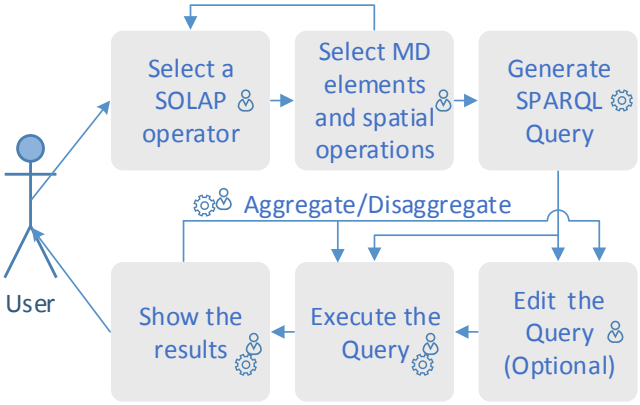


Fig. 17: Workflow diagram (Reproduced from [13])

ecuted. Finally, the results of the query are shown to the user (Fig. 18d). From this final step backward, the user may aggregate and disaggregate results or edit the query from the template and re-run again. Overview of GeoSemOLAP is given in Fig. 18. A screen-cast video on the actual use of GeoSemOLAP is also available online via our project site[32].

4.4 Demonstration and Discussion

Every MD element and spatial concept given at the high-level query "Total sales to customer grouped by city of their closest supplier" is S-ROLL-UP (Sales, [DISTANCE(Customer, Supplier)] → ClosestCity, SUM(SalesAmount)) are present in SPARQL terms as we highlighted and explained along in Section 4.1. This is achieved by mapping and annotating the spatial and MD elements explicitly with QB4SOLAP vocabulary in RDF terms. Once, the meta-data is in place implementing the SOLAP query generator algorithms to produce SPARQL queries for any given use case data is possible.

As a proof of concept, we have demonstrated GeoSemOLAP with the running use case example Geo-Northwind data set. A graphical and easy to understand representation of the use case data is given with GeoSemOLAP for users to easily interact with the system (Fig. 18a). Since our primary focus and contribution is around spatial concepts we have depicted only the *spatial* dimensions, *spatial* hierarchy levels, and *spatial* attributes, which are necessary for SOLAP operations. The center of the analysis is the Sales (fact cube) given with measures in orange boxes in the figure. Supplier and Customer dimensions are given in double-lined green ellipses, which also represent the base levels of the dimensions. The dimension hierarchy rolls-up to levels as

[32] http://extbi.cs.aau.dk/GeoSemOLAP

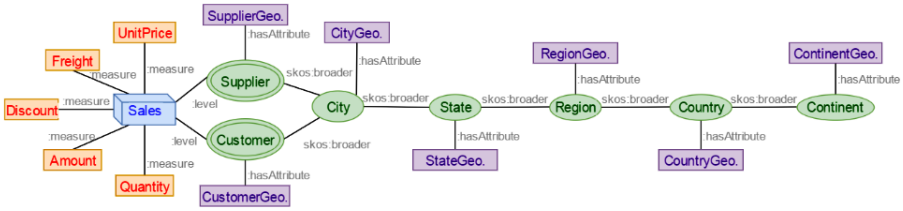
City, State, Region, Country, and Continent that are given in green single-line ellipses. Every level has spatial attributes given in purple boxes in the figure.

The example demonstrates the running query example *s-roll-up* (Ex. 8) on top of an *s-slice* as a nested query *(s-roll-up(s-slice(DW)))*. S-slice operator removes a dimension from a cube by choosing a single spatial level attribute. In the GUI of GeoSemOLAP, the user can specify the slice level by choosing the geometry of a level attribute through a map. Therefore, the operator comes with an option to select the geometry from a map (Fig. 18b). Next, the operator requires two spatial parameters: one parameter to specify the spatial location and another one to define the slice level with respect to the specified location. In the demo, we can see that the user clicked a location in Germany and specified the slice level as a country to make a projection on the sale observations in Germany (Fig. 18b).

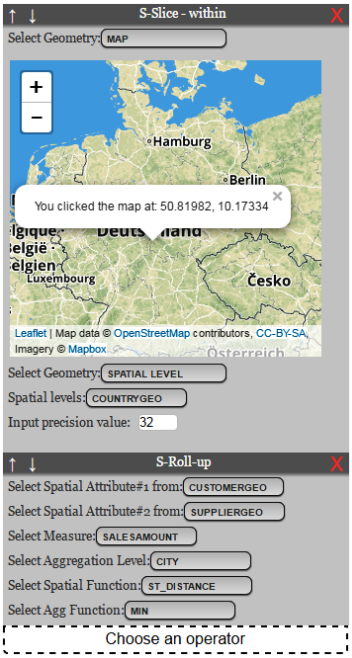
After the s-slice operator, an s-roll-up is added to the query by the user as seen in the bottom of Fig. 18b. Thus the user can aggregate the measures (total sales) and derive new perspectives in the selected location. We have chosen the same parameters and spatial functions to formulate the s-roll-up operator from the running use case query (Ex. 8). The overall demo depicts initially, an s-slice (at country level) on a location clicked on the map, which is Germany. This can be formulated in an intuitive way as “Project the sales (observations) at country level in Germany". Before adding an s-roll-up operator, the user can aggregate the measures for the chosen country level to see the general overview, which adds triple patterns to the SPARQL query to get aggregated values of all the measures (price, quantity, amount, discount, freight) as given in Fig. 19. After disaggregating and clearing the tables view in the interface, the s-roll-up can be added on top of the s-slice, which can be formulated as “Total sales to customers by the city of their closest suppliers *in Germany*". A sample snapshot of the results is given at the bottom of the figure (Fig. 18d).

By looking at the results in Fig. 18b, we can see that some customers in Germany have closer suppliers from other cities than Germany, such as Lyngby, which is in Denmark. This is a new pattern that SOLAP reveals dynamically by using spatial functions with OLAP operators interactively. With the help of GeoSemOLAP, now this kind of SOLAP operators can be easily formulated and mapped to SPARQL for querying the spatial RDF endpoints. GeoSemOLAP considerably breaks the limits of advanced spatial analysis on the SW for spatial data cubes. The only remaining limitation is that the spatial data cubes should be annotated and published on the SW by using QB4SOLAP vocabulary for GeoSemOLAP to be able to communicate with the spatial RDF endpoints and extract their metadata for query generation.

We have depicted our future vision of SOLAP on the Semantic Web in Section 3.5, Fig. 13, where an RDF2SOLAP module provisioned as a tool that can semi-automatically annotate the MD spatial RDF endpoints with QB4SOLAP,



(a) Graphical representation of an example use-case schema



(b) SOLAP operator configuration

(c) Generated SPARQL query for nested SOLAP

obs2	supplierCity	cityName	totalSales
http://qb4solap.org/cubes/instances/geonorthwind#sale_10643_3	http://qb4solap.org/cubes/instances/geonorthwind#city_22	"Lyngby"	18
http://qb4solap.org/cubes/instances/geonorthwind#sale_10323_2	http://qb4solap.org/cubes/instances/geonorthwind#city_6	"Berlin"	44.8
http://qb4solap.org/cubes/instances/geonorthwind#sale_10312_4	http://qb4solap.org/cubes/instances/geonorthwind#city_14	"Frankfurt"	62

(d) Example result for a nested SOLAP query (S-Roll-up (S-Slice()))

Fig. 18: Screenshot of GeoSemOLAP (Reproduced from [13])

which would limit the remaining barriers for end-users and business intelligence analysts to easily interact with semantic web data using SOLAP operations.

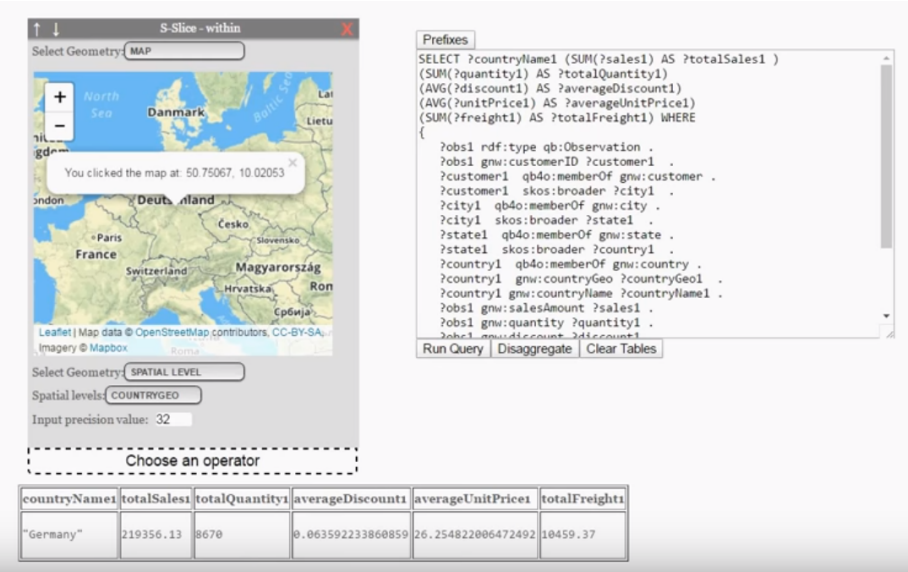


Fig. 19: Screenshot of s-slice projection and aggregation

5 Use Case: Spatial OLAP over Environmental and Farming Data with QB4SOLAP

This section gives an overview of Paper E [11].

5.1 Motivation and Problem Statement

In Section 2, we have explained that governmental organizations and agencies making huge amounts of data publicly accessible on the Semantic Web in Linked Open Data (LOD) format [3]. The Danish government is among one of those that releases its digital raw material as Open Data [5] starting from 2012. As an initial effort, GovAgriBus was published to SW in 2014[33], which contains governmental agricultural and business data sets with spatial information in LOD format. Therefore, the users can formulate interesting SPARQL queries across these domains including spatial functions and containment relationships. Moreover, we have confirmed that advanced analytical queries are also possible with the support of aggregate functions in SPARQL (Section 2). However, lack of well established multi-dimensional (MD) and spatial models had been an impediment to analyzing this kind of interesting LOD data with spatial OLAP, which is the prominent query technique in spatial data warehouses for business intelligence (BI) users and decision-makers.

[33] https://datahub.io/dataset/govagribus-denmark

Responding to SOLAP queries on the SW requires well-defined vocabularies and meta-models to facilitate spatial functions and OLAP operators in SOLAP.

Motivated by this need, QB4SOLAP [12, 15] is developed to support spatial MD analysis and SOLAP operations on the SW (Explained in Section 3). QB4SOLAP is tested on a non-trivial MD dataset with spatial concepts (Geo-Northwind, Fig. 10) to show how we can support interesting SPARQL queries on the SW by providing multi-dimensional and spatial context to SOLAP operators. However, QB4SOLAP has never been tested on complex real-world data sets published by governmental and public organizations. Since, this could lead to different challenges and problems, where corrupt data formats, complex spatial data types, and noisy data sets, etc. might be present in real-world data sets, we have decided to apply QB4SOLAP on a complex real-world data set from various domains including livestock farming and environmental data, where all domains contain spatial information of the data.

5.2 Spatial Data Cube of Livestock Holdings in Danish Farms

In the following, initially, we summarize the data sources for different domains. Raw data sets are retrieved in various formats from different governmental agencies in Denmark. Next, we give an overview of the spatial MD data cube, *GeoFarmHerdState*, which is modeled and created (in RDF format) with QB4SOLAP vocabulary by using data from the source data sets.

Source Data Sets

In order to find an interesting set of data from different resources, which has spatial information, we have investigated in the real-world problems in Danish farms. Danish Ministry of Environment regulates livestock units (LU or LSU)[34] per area in order to keep nitrate leaching under control in vulnerable areas, such as areas that have close proximity to drinking water sources and supply facilities or ammonia vulnerable areas [21]. Therefore, we find it particularly interesting to build our use case around the following domains listed in Table 9. Data source and approximate size of the data sets (as a number of records) for each domain is given in the table. The data format of the data sets from these three domains is *Shape* file, with geographical coordinates.

[34]Livestock unit is a reference unit that facilitates the aggregation of livestock from various species [11]. It is simply used to produce statistics describing the number of livestock on farms. https://ec.europa.eu/eurostat/statistics-explained/index.php/Glossary:Livestock_unit_(LSU)

Table 9: Use case data domains and sources

Domain	Source	Size
Livestock Farming Data	http://jordbruganalyser.dk	240,000 records
Environmental Data	http://www.miljoeportal.dk	30,000 records
Spatial Data	http://www.geodata-info.dk	2300 records

Livestock Farming (CHR) Data. The Ministry of Environment and Food of Denmark is responsible for maintaining and publishing the livestock farming central database. This database contains *central husbandry* (livestock) *registry* (CHR) data. The database is updated yearly. We have downloaded the data for around 40.000 farms, where the state of the farms recorded between the years 2010-2015. Some interesting attributes in this data set are CHR number, CVR number (Central Company Registry), LSU/LU (Livestock unit), address of the farm, coordinates of the farm, types of the herd, number of animals (per herd), animal usage code and purpose, etc.

Environmental Data. Denmark's environment portal hosts public environmental data about soil quality, vulnerable sites, nitrate catchment areas. We have downloaded three different data sets containing information about *nitrogen reduction potentials, phosphor, and nitrate classifications* of the soil in the whole of Denmark. The measurements of the soil contain data from 2008 to 2015. The most interesting attributes from there data set are Nitrogen reduction potentials in the areas, nitrate classification type, and phosphor classification type.

Spatial Data. In order to build a spatial data cube and enrich the analysis, we have added geographical data sets to our use case, which contains *parish* and *drainage areas* of Denmark. Some attributes from these data sets are the Drainage area name and code, total area, parish name, and ID, etc.

GeoFarmHerdState Cube in RDF

By using the afro-mentioned use case data sets from spatial, environmental, and livestock farming domains, we have created a spatial data cube of livestock holdings that we refer to as *GeoFarmHerdState*. The data cube is created after thoroughly analyzing the interesting attribute columns (from the Shape files) and joining them on referential integrity constraints or by using spatial joins through the geographical coordinates. The highlights and challenges of the transformation and conciliation process for generating GeoFarmHerdState spatial data cube in RDF are discussed in Section 5.4.

We have used QB4SOLAP [12, 15] vocabulary for defining the GeoFarmHerdState cube *schema* and data cube *instance* members in RDF. GeoFarmHerd-

State schema contains the meta-data, which is used to describe spatial and MD concepts such as spatial dimensions, spatial hierarchies, spatial attributes, and measures. GeoFarmHerdState data cube instances are the members of the cube schema, such as level and attribute members, fact members, and measure values, which represent the actual data records. Fig. 20 depicts a conceptual schema of GeoFarmHerdState data cube. In the following, we give examples of the cube schema concepts in RDF terms with QB4SOLAP.

GeoFarmHerdState Cube schema concepts. A cube schema is described with multi-dimensional concepts of data warehouses such as Dimensions, Levels, Attributes, Hierarchies, Hierarchy steps, and Measures. A spatially extended cube schema has spatial extensions for all these concepts. All these concepts for spatially extended data warehouses can be annotated with QB4SOLAP vocabulary. The examples for these concepts are given below, reproduced from Paper E.

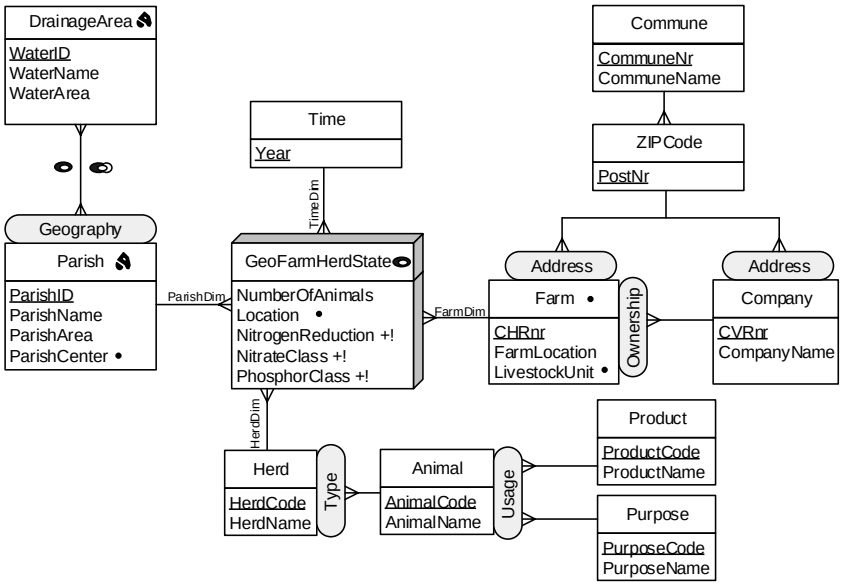


Fig. 20: GeoFarmHerdState – Conceptual MD schema of livestock holdings data (Adapted from [11])

Example 9 ((Spatial) Dimensions - Adapted from [11])
The MD conceptual schema in Fig. 20 has four dimensions: *Herd,Time, Parish,* and, *Farm*. Two of them (Farm and Parish) are spatial dimensions and given in RDF terms as follows:

```
gfs:farmDim rdf:type qb:DimensionProperty ; qb4o:hasHierarchy gfs:ownership , gfs:address .
gfs:parishDim rdf:type qb:dimensionProperty ; qb4o:hasHierarchy gnw:geography .
```

A dimension is a *spatial* dimension, if it has at least one spatial level. Farm (base) level is a spatial level in Farm dimension since farm has location as coordinates, and Parish (base) level and DrainageArea level are spatial levels (with polygon coordinates) in Parish dimension.

Example 10 ((Spatial) Hierarchies - Adapted from [11])
Hierarchies are found in dimensions as given in Ex. 9 with `qb4o:hasHierarchy` property. Hierarchies are composed of levels, and a hierarchy is *spatial*, if it has at least one spatial level. In the following we give three hierarchies *Geography, Usage,* and, *Address* from GeoFarmHerd-State cube, which are given in ellipses Fig. 20).

```
gfs:geogprahy rdf:type qb4o:Hierarchy ; qb4o:inDimension gfs:parishDim ;
    qb4o:hasLevel gfs:drainageArea .
gfs:usage rdf:type qb4o:Hierarchy ; qb4o:inDimension gfs:herdDim ;
    qb4o:hasLevel gfs:product , gfs:purpose .
gfs:address rdf:type qb4o:Hierarchy ; qb4o:inDimension gfs:farmDim ;
    qb4o:hasLevel gfs:zipCode , gfs:commune .
```

The Geography hierarchy is a *non-strict* spatial hierarchy. A spatial hiearchy is non-strict if it has at least one $(n-n)$ relationship between its levels. In the Geography hierarchy (Fig. 20) the $(n-n)$ cardinality represents that a parish may belong to more than one drainage area. Usually, non-strict spatial hierarchies arise when a partial containment relationship exists, which is given as *Intersects* in our use case. Usage hierarchy is a *generalized* hierarchy with non-exclusive paths to splitting levels (Product and Purpose) and has no joining level but the top level *All*. Finally, the Address and Ownership hierarchies are *parallel dependent* hierarchies. Parallel hierarchies arise when a dimension has several hierarchies sharing some levels. Note that the Address hierarchy has different paths from the Company and Farm levels (Fig. 20) [11].

Example 11 ((Spatial) Levels - Adapted from [11])
We present a spatial level (Parish) as an example with its attribute (name)s. Attributes and spatial attributes of levels are given in Ex. 12. A level is *spatial*, if it has an associated *geometry*. Parish spatial level has an associated geometry polygon given with `geo:hasGeometry` property. Some other spatial characteristics of the levels can be recorded in the spatial attributes of the level such as the center point of the parish (`gfs:parishCenter`) as given in the next example (Ex. 12).

```
gfs:parish rdf:type qb4o:LevelProperty ; qb4o:hasAttribute gfs:parishID ;
    qb4o:hasAttribute gfs:parishName ; qb4o:hasAttribute gfs:parishArea ;
    qb4o:hasAttribute gfs:parishCenter ; geo:hasGeometry gfs:parishPolygon.
```

Example 12 (Spatial and non-spatial level attributes - Adapted from [11]) Attributes keep characteristics of the level as a value, i.e., in terms of string or literals. *Spatial* attributes (and attribute values) are defined over a spatial domain. For example, non-spatial attributes are defined as ranging over XSD literals[35] where spatial attributes must be ranging over spatial literals, i.e., well-known text literals (WKT) from OGC schemas[36]. Spatial attributes are a sub-property of the `geo:Geometry` class. Further, the domain of the spatial attribute should be specified with `rdfs:domain`, which must be a geometry. Finally, the spatial attribute must be specified as an instance of `geo:SpatialObject` with the `rdfs:subClassOf` property. Examples of attributes are given in the following. The example below shows the spatial and non-spatial attributes of the Parish level.

```
gfs:parishID rdf:type qb4o:LevelAttribute ; qb4o:inLevel gfs:parish ;
    rdfs:range xsd:positiveInteger .
gfs:parishName rdf:type qb4o:LevelAttribute ; qb4o:inLevel gfs:parish ;
    rdfs:range xsd:string .
gfs:parishCenter rdf:type qb4o:LevelAttribute ; rdfs:subPropertyOf geo:Geometry ;
    qb4o:inLevel gfs:parish ; rdfs:domain geo:Point; rdfs:subClassOf geo:SpatialObject;
    rdfs:range geo:wktLiteral , virtrdf:Geometry .
```

In Ex. 11, it is mentioned that spatial levels are defined through their associated geometries, which are not given as a level attribute. For the Parish level, we present the following example of the corresponding geometry.

```
gfs:parishPolygon rdf:type geo:Geometry; rdfs:domain geo:MultiSurface;
    rdfs:subClassOf geo:SpatialObject; rdfs:range geo:wktLiteral , virtrdf:Geometry .
```

Example 13 (Spatial Hierarchy steps - Adapted from [11]) Hierarchy steps define the structure of the hierarchy in relation to its corresponding, levels. A hierarchy step entails a roll-up relation between a lower (child) level and an upper (parent) level with a cardinality. The cardinality $(n-n, 1-n, n-1, n-n)$ relationship describes the number of members in one level that can be related to a member in the other level for both child and parent levels. A hierarchy step is *spatial* if it relates a spatial child level and a spatial parent level, in which case it entails a topological relationship between these spatial levels. Both spatial and non-spatial hierarchy steps are defined as a blank node with the `qb4o:HierarchyStep` property and linked to their hierarchies with the `qb4o:inHierarchy` property. The parent and child levels are linked to hierarchy steps with the

`qb4o:childLevel` property and the `qb4o:parentLevel` property. The cardinality of a hierarchy step is defined by the `qb4o:pcCardinality` property. And finally, the topological relationship of a hierarchy step is defined by the `qb4so:pcTopoRel` property.

The following illustrates the hierarchy steps of the spatial hierarchy Geography and non-spatial hierarchy Address, which has different paths from child levels Farm and Company.

```
## Geography hierarchy structure ##
_:geography_hs1 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:geography ;
    qb4o:childLevel gfs:parish ; qb4o:parentLevel gfs:drainageArea ;
    qb4o:pcCardinality qb4o:ManyToMany; qb4so:pcTopoRel qb4so:Intersects, qb4so:Within .
## Address hierarchy structure ##
_:farm_address_hs1 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:farm ; qb4o:parentLevel gfs:zipCode ;
    qb4o:pcCardinality qb4o:ManyToOne .
_:farm_address_hs2 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:zipCode ; qb4o:parentLevel gfs:commune ;
    qb4o:pcCardinality qb4o:ManyToOne .
_:company_address_hs1 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:company ; qb4o:parentLevel gfs:zipCode ;
    qb4o:pcCardinality qb4o:ManyToOne .
_:company_address_hs2 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:zipCode ; qb4o:parentLevel gfs:commune ;
    qb4o:pcCardinality qb4o:ManyToOne .
```

Example 14 (Spatial and non-spatial measures - Adapted from [11])
Measures record the values of a phenomenon being observed. Measures and *spatial* measures are defined with `qb:MeasureProperty`. A measure is spatial if it is defined over a spatial domain. Similarly to attributes (Ex. 12), measures are defined ranging over XSD literals and spatial measures must be ranging over spatial literals. The following shows an example of a spatial measure (Location) and a non-spatial measure (NumberOfAnimals).

```
gfs:location rdf:type qb:MeasureProperty ; rdfs:subPropertyOf sdmx-measure:obsValue ;
    rdfs:subClassOf geo:SpatialObject ; rdfs:domain geo:Point ;
    rdfs:range geo:wktLiteral , virtrdf:Geometry .
gfs:numberOfAnimals rdf:type qb:MeasureProperty ;
    rdfs:subPropertyOf sdmx-measure:obsValue ; rdfs:range xsd:decimal .
```

Example 15 (Fact schema - Adapted from [11])
In a fact schema, the data structure (DSD) of the cube is annotated with `qb:DataStructureDefinition`. The dimensions are given as `components` and defined with the `qb4o:level` property as the dimensions are linked to the fact at the lowest granularity level. A fact is *spatial* if it relates two or more spatial levels. Similarly, measures are given as `components` of the

fact and are defined with the `qb:measure` property. Aggregation functions on measures and spatial aggregation functions on spatial measures are also defined in the DSD with `qb4o:aggregateFunction`. Fact-level cardinality relationships and topological relationships are defined with `qb4o:cardinality` and `qb4so:topologicalRelation` in DSD. The following shows the data structure definition of the cube GeoFarmHerdState, which is defined with corresponding measures and dimensions.

```
# – GeoFarmHerdState Cube Definition of the Fact FarmHerdState
gfs:GeoFarmHerdState rdf:type qb:DataStructureDefinition ;
    # Lowest level for each dimensions in the cube
    qb:component [qb4o:level gfs:herd ; qb4o:cardinality qb4o:ManyToOne ] ;
    qb:component [qb4o:level gfs:time ; qb4o:cardinality qb4o:ManyToOne ] ;
    qb:component [qb4o:level gfs:farm ; qb4o:cardinality qb4o:ManyToOne ;
        qb4so:topologicalRelation qb4so:Equals ] ;
    qb:component [qb4o:level gfs:parish ; qb4o:cardinality qb4o:ManyToMany ;
        qb4so:topologicalRelation qb4so:Within ] ;
    # Measures in the cube
    qb:component [qb:measure gfs:numberOfAnimals ; qb4o:aggregateFunction qb4o:Sum] ;
    qb:component [qb:measure gfs:location ; qb4o:aggregateFunction qb4so:ConvexHull] ;
    qb:component [qb:measure gfs:nitrogenReduction ; qb4o:aggregateFunction qb4o:Avg] ;
    qb:component [qb:measure gfs:nitrateClass ; qb4o:aggregateFunction qb4o:Avg] ;
    qb:component [qb:measure gfs:phosphorClass ; qb4o:aggregateFunction qb4o:Avg] .
```

GeoFarmHerdState Cube instance members. Remark the definition from Section 3, where we distinguished the MD data cube elements by two definitions levels for defining with QB4SOLAP at the schema level and annotating with QB4SOLAP at the instance level. GeoFarmHerdState Cube schema concepts are defined with QB4SOLAP along with the Examples 9-15 given above. Fact members (with measure values) and Level members (with attribute values) are annotated with QB4SOLAP and exemplified as GeoFarmHerdState cube instance members in the following.

Example 16 (Fact members - Adapted from [11])
Fact members (i.e., facts of GeoFarmHerdState) are instances of the `qb:Observation` class. Each fact member is related to a set of dimension *base* level members and has a set of measure values. Every fact member has a unique identifier (IRI) which is prefixed with `gfsi:`.

The following shows an example of a single fact member, which represents the state of a farm with CHR no. 39679 in the year 2015 that has the herd code 15.

```
gfsi:farm_39679_2015 rdf:type qb:Observation ;
## Dimension levels and base level members associated with the fact member
    gfs:herdCode gfsi:herd_15 ; gfs:year gfsi:year_2015 ;
    gfs:chrNumber gfsi:farm_39679 ; gfs:parishID gfsi:parish_8311 ;
## Measures associated with the fact member
    gfs:numberOfAnimals "100.0"^^xsd:decimal ; gfs:nitrateClass "3"^^xsd:integer ;
```

```
gfs:nitrogenReduction "0.75"^^xsd:decimal ; gfs:phosporClass "3"^^xsd:integer ;
gfs:location "POINT(8.3713 56.7912)"^^geo:wktLiteral .
```

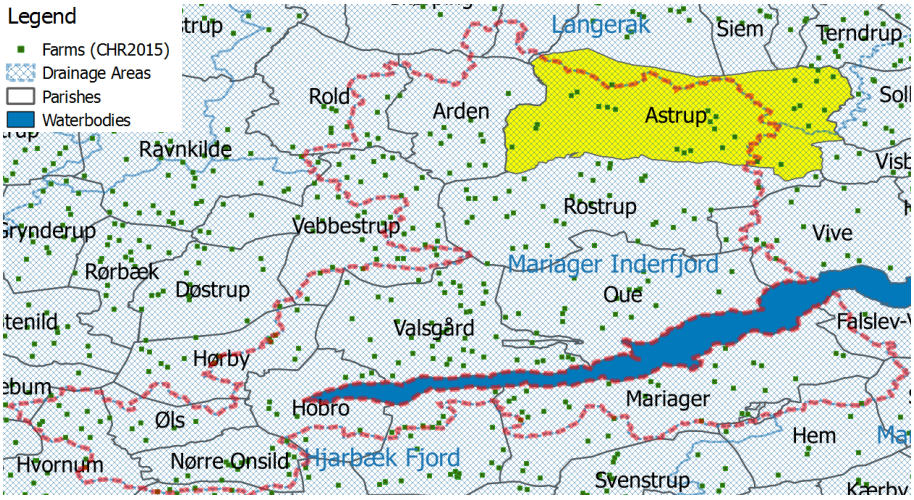


Fig. 21: GeoFarmHerdState – Fact members and Level members of Ex. 17 marked (Adapted from [11])

Example 17 (Level Members - Adapted from [11])
Level members are defined with `qb4o:LevelMember`. They are linked to their corresponding levels from the schema with the `qb4o:memberOf` property. For each level member there is a set of attribute values. Due to the roll-up relations between levels of hierarchy steps (Ex 13),the `skos:broader` property relates a child level member to its parent level member. For spatial level members in QB4SOLAP, we extend `skos:broader` relations with explicit topological relations, which describes precisely the nature of spatial relation between two level members, e.g., `qb4so:intersects` or `qb4so:within`.

The following shows an example of a child level member in the Parish level and one of its parent level members in the DrainageArea level from the Geography dimension. Fig. 21 presents a map snapshot for fact members and level members. Parish level member “Astrup" is highlighted and DrainageArea level member “Mariager Inderfjord" is marked with red borders. Note that Astrup intersects with another drainage area “Langerak", therefore it links to two different parent level members with `skos:broader` property. We have implicitly given the topological relation between the level members with `qb4so:intersects` predicate for the parish level member (1) and its parent level members in the blue line. Similarly, parish

level member (2) - Oue is given with an implicit topological relation `qb4so:within` to its parent level member 'Mariager Inderfjord in the blue line of the listing.

```
## Parish level member (1) Astrup
gfsi:parish_8460 rdf:type gfs:parish ;
    qb4o:memberOf gfs:parish ; skos:broader gfsi:water_3710, gfsi:water_159 ;
    qb4so:intersects gfsi:water_3710, gfsi:water_159 ;
    gfs:parishID 8460 ; gfs:parishName "Astrup" ; gfs:parishArea "28857338.30518"^^xsd:double ;
    gfs:parishCenter "POINT(10.095657 57.476879)"^^geo:wktLiteral ;
    gfs:parishPolygon "POLYGON((10.4038 56.7963, 10.3984 56.7721, 10.3411 56.7372, 8.3078
    56.7281, 10.2987 56.7601, 10.2563 56.7763, 10.3511 56.8137, 10.4038 56.7963))"^^geo:wktLiteral .
## Parish level member(2) Oue
gfsi:parish_8309 rdf:type gfs:parish ;
    qb4o:memberOf gfs:parish ; skos:broader gfsi:water_159 ;
    qb4so:within gfsi:water_159 ;
    gfs:parishID 8309 ; gfs:parishName "Oue" ; gfs:parishArea "33297796.91284"^^xsd:double ;
    gfs:parishCenter "POINT(8.2552, 56.8176)"^^geo:wktLiteral ;
    gfs:parishPolygon "POLYGON((8.4038 56.7963, 8.3984 56.7721, 8.3411 56.7372, 8.3078
    56.7281, 8.2987 56.7601, 8.2563 56.7763, 8.3511 56.8137, 8.4038 56.7963))"^^geo:wktLiteral .
## DrainageArea level member
gfsi:water_159 rdf:type gfs:drainageArea ;
    qb4o:memberOf gfs:drainageArea ; gfs:waterID 159 ;
    gfs:waterName "Mariager Inderfjord" ; gfs:waterArea 267,477 ;
    gfs:drainageGeo "POLYGON((8.6048 56.9843, 8.5908 56.8969, 8.5707 56.8664,
    8.5975 56.8519, 8.5215 56.8483, 8.3959 56.7625, 8.3938, 56.7340, 8.3613 56.6802,
    8.2584 56.7764, 8.2475 56.7051, 8.2175 56.7232, 8.3121 56.8441, 8.2806 56.8659,
    8.3602 56.9569, 8.4786 56.9713, 8.5474 56.9905, 8.6048 56.9843))"^^geo:wktLiteral .
```

5.3 SOLAP examples over GeoFarmHerdState in SPARQL

Spatial OLAP - SOLAP can be applied to spatial data cubes and helps users to query and analyze data with enhanced spatial capabilities of OLAP by benefiting from the spatial information in the spatial data cube. Principally, a SOLAP operation should include a spatial condition or a spatial function (Sect. 3.3, SOLAP Operations). Spatial conditions are spatial Boolean predicates that define spatial constraints on the geometries of spatial cube level member attributes and measures. These can be interpreted as topological relations where the relations between two spatial geometry objects are defined topologically and the result of topological relation/spatial Boolean predicate is binary (True/False). On the other hand, spatial functions can be spatial numerical operations (e.g., distance, area) or spatial aggregate functions (e.g., spatial union, intersection) and return new data from the cube members. These operations can be used to build dynamic spatial hierarchies in SOLAP operations.

Finally, we present two common SOLAP operators (s-dice and s-roll-up) in the following with non-trivial and interesting query examples on the GeoFarmHerdState cube.

Example 18 (S-Dice - Adapted from [11]))
S-dice operator keeps the cells of the cube that satisfy the spatial predicate over dimension levels, attributes, or measures. It returns a subset of the cube with filtered members of the cube.

The following items and listings show two different s-dice operator examples by filtering with a spatial predicate on levels and measures.
1. Filter the farms located within 5 km buffer from the center of a drainage area.
2. Filter the farms located within 2 km distance from the center of their parish, which is in the nitrate class I areas.

```
# 1 - s-dice on dimension levels #
SELECT ?obs WHERE {
    ?obs rdf:type qb:Observation ;
       gfs:farmID ?farm ;
       gfs:parishID ?parish .
    ?farm gfs:farmLocation ?farmGeo .
    ?parish qb4o:memberOf gfs:parish ;
       skos:broader ?drainageArea .
    ?drainageArea gfs:waterPolygon ?drainagePoly .
BIND (bif:st_centroid (?drainagePoly) as ?drainageCenter)
FILTER (bif:st_within(?drainageCenter, ?farmGeo, 5)) }
```

```
# 2 - s-dice on measures #
SELECT ?obs WHERE {
    ?obs rdf:type qb:Observation ;
       gfs:location ?farmLocation ;
       gfs:nitrateClass ?nitClass ;
       gfs:parishID ?parish.
    ?parish gfs:parishCenter ?parishCent .
BIND (bif:st_distance (?farmLocation, ?parishCent)
AS ?distance)
FILTER (?distance < 2 && ?nitClass = 1)}
```

The first s-dice operation uses a spatial function that is applied on level members of the DrainageArea level to get the *center* of their polygon geometries. The level members of the Farm level are filtered with a spatial Boolean predicate with respect to the farm locations that are *within* a 5 km buffer area of the center of the drainage areas.

The second s-dice operation uses a spatial function that is applied to the spatial measure - farm location, in order to get the *distance* of the farms from the center of their parish, which is followed by two Boolean predicates; 1) to filter the farms that are less than 2 km away from the center of their parishes and 2) filter the farms that are on nitrate class I areas.

Example 19 (S-Roll-up - Adapted from [11]))
S-roll-up operator aggregates measures of a given cube by using an aggregate function and a spatial function (or a spatial predicate) along a spatial dimension's hierarchy. It returns a cube with measures at a coarser granularity for a given dimension.

In the following, we present two examples of the s-roll-up operator.
1. The total amount of animals on the farms, which are closest to their parishes' center.
2. The average percentage of nitrogen reduction potentials in the parishes that are within and/or intersect the drainage area "Nibe-Bredning".

```
# 1 - s-roll-up #
SELECT ?parish (SUM(?animalCount) AS ?totalAnimals)
WHERE { ?obs rdf:type qb:Observation ;
    gfs:numberOfAnimals ?animalCount;
    gfs:farmID ?farm ;
    gfs:parishID ?parish .
?farm gfs:farmLocation ?farmGeo .
?parish gfs:parishCenter ?parishCent .
# Inner select for finding the
# closest farms to the parish centers #
    {SELECT ?farm1 (MIN(?distance) AS
    ?minDistance) WHERE
    { ?obs rdf:type ab:Observation ;
       gfs:farmID ?farm1;
       gfs:parishID ?parish1 .
    ?farm1 gfs:farmLocation ?farm1Geo .
    ?parish1 gfs:parishCenter ?parish1Cent.
    BIND (bif:st_distance (?farm1Geo, parish1Cent)
    AS ?distance) } GROUP BY ?farm1 }
    FILTER (?farm = ?farm1 && bif:st_distance
    (?farmGeo, ?parishCent) = ?minDistance )}
GROUP BY ?parish

# 2 - s-roll-up #
SELECT ?drainageArea (AVG(?nitRed) AS ?avgNitRed)
WHERE { ?obs rdf:type qb:Observation ;
    gfs:location ?farmLocation ;
    gfs:nitrogenReduction ?nitRed ;
    gfs:parishID ?parish.
    ?parish qb4o:memberOf gfs:parish ;
       gfs:parishPolygon ?parishGeo ;
       skos:broader ?drainageArea .
    ?drainageArea gfs:memberOf gfs:drainageArea ;
       gfs:waterPolygon ?drainageGeo ;
       gfs:waterName ?drainageName .
```

```
FILTER (bif:st_within(?parishGeo, ?drainageGeo)
|| bif:st_intersects(?parishGeo, ?drainageGeo)
&& ?drainageName = "Nibe-Bredning")}
GROUP BY ?drainageArea
```

In the first s-roll-up operator, measures are aggregated to the Parish level after selecting the farms with respect to their proximity to the center of the parish with a spatial function. This creates a dynamic spatial hierarchy by defining the aggregation level members of the parent level (parish) by the proximity of the child level members (farms) to the center of the parent level's geometry.

In the second s-roll-up operator, measures are aggregated to a specified drainage area ("Nibe-Bredning") at the DrainageArea level. We select all the possible topological cases where a parish intersects or within the drainage area.

5.4 Discussion and Perspectives

In order to demonstrate that QB4SOLAP can be applied to a complex real-world use case, we have selected various interesting domains (i.e., geographical data, farming, and environmental data) to link, relate, conciliate and annotate cross-domains with QB4SOLAP. To achieve our goal we come across several challenges that we discuss and give our perspectives in the following steps.

Data specification, analysis, and modeling process

Initially, we had to find interesting and relatable domains and specify the data scope of the data in these domains by analyzing the available data sources. We have given the data sources and explained our use case domains of livestock farming (CHR) data, environmental data, and spatial data in Sect. 5.2. To find correct environmental and spatial relations across these data sets require certain domain knowledge and expertise. Therefore, understanding the domain interests, specifying the domain interests, and a comprehensive pre-analysis was required steps before starting modeling our use case schema. We pursued comprehensive research from domain resources ([22] and [21]) to shape our use case GeoFarmHerdState. For example, we have decided the center of analysis as farms and the number of livestock in farms, together with soil quality measurements of the farms. This decision is made with respect to domain interest for the analysis of nitrate leaching to ground and surface water from livestock farming [22].

In order to provide insight into these analyses, we have also decided to include animal herd type and usage of animal products as non-spatial dimensions and hierarchies. In our spatial data domain, we have selected parishes and (groundwater) drainage areas as geographical areas to build our spatial hierarchy. This means that farm measures and statistics can be aggregated and analyzed along with parish and drainage area levels. We have not built up a geographical hierarchy with farm cities, communes, and regions since the domain interest is not to analyze livestock farming effects on soil quality at the city level or region level but directly at the groundwater and drainage area level. Parish (church regions) has been selected as the intermediary level between farms and drainage areas, since administratively and historically farms are related to parishes but not directly to communes or cities.

After getting domain knowledge and specifying the scope of our data sets and domain, we conceptually modeled a spatial data cube - GeoFarmHerdState by using the principals of spatially extended MultiDim models [32]. The final conceptual MD schema of the use case is depicted in Fig. 20. QB4SOLAP vocabulary allows us to annotate spatially extended MultiDim conceptual models, and it supports state-of-the-art semantic spatial data cubes. By using QB4SOLAP, we can annotate all the spatial cube concepts such as spatial measures, spatial hierarchies, topological relations at spatial hierarchy steps, etc. Therefore, we decided to use QB4SOLAP for transforming the use case data and generating in RDF format in order to publish on the Semantic Web, in the following steps.

Data conciliation, transformation, and generation

The line of tasks in this step involves the technical process of RDF data generation. Since we use data sets from different domains in different formats, our starting point was to relate and conciliate data sets by using the unique identifiers (if they are available), in order to create a relational implementation of the GeoFarmHerdState spatial data cube. For example, CHR number, CVR number, Postal number are unique identifiers that can be used to relate different tables. On occasions where there are no unique identifiers are available, we utilized spatial joins by overlaying spatial coordinates of different datasets with a GIS tool, and joining attributes from one geometry feature to other features, based on the containment relationships. For example, we used spatial joins to get soil quality attributes of farmlands, by intersecting the farm location data set point coordinates with environmental data set polygon coordinates, where we derived soil quality measurements for nitrogen reduction potentials, phosphor and nitrate classifications of each farm. We have also used spatial joins for building up the spatial hierarchy between Parishes and Drainage areas by overlaying parish and drainage area polygons. This way we can find the exact parishes that are related to the

exact drainage areas as child-parent level members in a hierarchy step, and annotate them accordingly. Moreover, we can as well find the exact relationship in topological terms i.e., intersects, within to explicitly annotate the relation. Since, there is an $(n - n)$ cardinality, between parishes and drainage areas, where a parish may intersect with more than one drainage area and vice-versa, it is important to know the exact relation between level members of the hierarchy.

Relating the data sets via referential integrity keys are done after importing raw data in an RDBMS. All of the spatial data conciliation steps are pursued in a GIS tool. Once we have derived new spatial containment relationships, we have also imported the spatial data into the RDBMS with spatial support.

After conciliating the data sets from different domains with respect to the conceptual model of the GeoFarmHerdState cube given in Figure 20, we exported 12 tables (in CSV format) as the relational representation of an MD cube schema such as snowflake schema, which is the tabular cube form of GeoFarmHerdState. The tables are composed of: *one* fact table (GeoFarmHerdState) with measure values and foreign keys of the dimensions base levels, *four* dimension base level tables where *two* of them are spatial dimensions, and *seven* tables for the remaining levels along the hierarchies of the four dimensions. The (level) tables along the same hierarchy are also related to each other with referential integrity constraints. The tables contain also level attributes and attribute values describing the characteristics of the level members. The final RDF files are generated with ad-hoc C# code mapping the CSV files into RDF by using the relations and QB4SOLAP vocabulary annotations.

Data publication and exploitation process

The final prepared RDF files are published to a public endpoint by using a triple store of our choice. We have chosen the Virtuoso Open Source Universal Server (Version 07.20.3217) as a triple store.

The main goal of this paper was to demonstrate how we can re-use open spatial government data and enrich the data modeling with QB4SOLAP by publishing a spatial multi-dimensional data set and querying with SOLAP operators. In order to make it easier for the users and readers, we have given the SPARQL endpoint and example queries on our project page[37].

In our SOLAP query example (Ex. 19), there might be two possible issues with the second SOLAP query[38]. *The first one is*, we use two built-in functions for finding the topological relations (`bif:st_instersects` and

[37] `http://extbi.cs.aau.dk/GeoFarmHerdState`

[38] **Remark Query**: Average percentage of nitrogen reduction potentials (measure in farm fact member) in the parishes (child level members in spatial hierarchy) that are *within or intersect* (topological relations) the drainage area Nibe-Bredning (parent level member in spatial hierarchy).

`bif:st_within`) between parish and drainage area level members. The built-in functions depend on the spatial capability of the chosen triple store. Even though some of the triple stores support spatial geometries and functions to an extent, it is not very efficient to process queries with spatial Boolean predicates between each level members during query processing time due to missing spatial indexes and complexity of spatial geometries such as polygons and multi-polygons. In order to prevent this, the RDF data can be published with explicit topological relations in QB4SOLAP using `qb4so:within` and `qb4so:intersects` predicates, then the filter clause in the SOLAP query can refer to these predicates for finding the topological cases.

The second issue is that the query selects all the possible topological cases, where a parish intersects or within the drainage area (Nibe-Bredning) to aggregate the measure (nitrogen reduction potentials), which is annotated at the granularity of the fact members (farms). This means: when a parish member intersects with the drainage area (Nibe-Bredning) measures for the farms that are outside Nibe-Bredning are also aggregated to this drainage area. In order to prevent this, the query should also include an s-drill-down operator to farm fact members from parish members with the spatial Boolean predicate `within` the drainage area (Nibe-Bredning), and aggregate the measures for the cases satisfying the predicate. Or ideally, the relation between fact members (farms) and drainage area (parent level members) can be annotated with QB4SOLAP before publishing, where the predicate for the relationship can be also included in the filter clause of the SOLAP query.

Since, it is very common to query data warehouses with nested (spatial) OLAP queries, in order to fully exploit GeoFarmHerdState data cube an elaborated nested SOLAP query can be created by selecting a sub-cube with *s-dice*, which can be followed with an *s-slice* on top of an *s-roll-up*.

For example, the pattern $({}^{3}\textit{s–roll–up}({}^{2}\textit{s–slice}({}^{1}\textit{s–dice}(\textit{GeoFarmHerdState}))))$ represent a typical nested SOLAP operation that can be paraphrased for the use case GeoFarmHerdstate as in the following: [1]Filter the farm states located within a 2 km distance from the center of their parish and [2]slice on the parish which has the most number of topological relations (intersects, within) with a drainage area, [3]average the nitrogen reduction potential of the drainage areas intersecting with the parish (Adapted from [11]).

Final remarks on conclusion and future work

GeoFarmHerdState data cube as a QB4SOLAP vocabulary use case demonstrates how interesting topical domains from governmental and spatial data sets can come together, which are modeled in a multi-dimensional way, in order to be published on the Semantic Web for enabling SOLAP queries (in SPARQL) on the Semantic Web. GeoSemOLAP tool (Section 4) already provides a GUI for data warehouse users to perform high-level SOLAP opera-

tions that can be translated into SPARQL to query MD spatial Semantic Web data.

The issues we come across during data conciliation and data exploitation process inspired us with new challenges to handle within the scope of this research. An automated enrichment module that can automatically detect MD and spatial concepts on existing Semantic Web data and annotate these concepts with QB4SOLAP can easily be a significant contribution to make the existing RDF endpoints ready to query with SOLAP operators.

6 Spatial Enrichment of Semantic Web Cubes with RDF2SOLAP Framework

This section gives an overview of Paper F [14]

6.1 Motivation and Problem Statement

The future vision of SOLAP was given in Section 3, which was depicted in Fig. 13. In that line of work, we have built the foundation for Geo-semantic (RDF) data warehouses with QB4SOLAP vocabulary and SOLAP operators in SPARQL query language, and provided GeoSemOLAP [13] that can automatically generate complex SPARQL queries from non-trivial nested SOLAP operators. In order to utilize GeoSemOLAP, RDF data on the Semantic Web is required to be annotated with QB4SOLAP. In the current state of the Semantic Web, there are many multi-dimensional data sets with spatial information that are published with RDF Data Cube (QB) Vocabulary [34]. QB4OLAP vocabulary addresses the MD modeling challenges of the QB Vocabulary [10] and complies with tools like QB2OLAP enrichment module (QB2OLAPem) [33], which can enrich and annotate the existing RDF QB endpoints with QB4OLAP that allows users to query the RDF data with traditional OLAP operators. However, if a data warehouse user would like to query the existing RDF data (containing natively spatial information) with spatial OLAP (SOLAP) operators, it has until now required to migrate the data into a traditional spatial data warehouse with spatial annotations on the MD concepts. This migration process is labor-intensive and not a preferable approach, as it removes the data from the Semantic Web and keeps the data in a closed, proprietary format of the in-house (spatial) data warehouse. (Semi-)Automatic Spatial enrichment of the existing Semantic Web cubes (QB, QB4OLAP) with QB4SOLAP Vocabulary, can minimize the user effort for querying the RDF data with SOLAP operators. Therefore, an established way of QB4SOLAP annotation from the existing RDF endpoints is required with consideration of existing SW technologies and tools. In

this line of work, enrichment algorithms for different scenarios and a tool (RDF2SOLAP) to implement the algorithms are proposed.

6.2 Enrichment Approach

The spatial enrichment approach employs GeoFarmHerdState use-case (from Section 5) as a proof of concept. We only focus on the spatial concepts of the use case, which is given as a remark in Figure 22 without the non-spatial dimensions from Figure 20. As can be seen on the figure, GeoFarmHerdState cube has two spatial dimensions: *Parish* and *Farm*. Parish dimension has a spatial hierarchy (*Geography*), which is composed of two spatial levels: *Parish* level and *DrainageArea* level. Farm dimension does not have a spatial hierarchy and have a single spatial level: *Farm*. *GeoFarmHerdState* fact is the center of analysis that represents the state of the farms for a certain time period (via dimension *Time*) with measures such as: *NumberOfAnimals* on the farm, *NitrogenReductionPotentials* of the farm land (soil), and *FarmLocation* (which is a spatial measure).

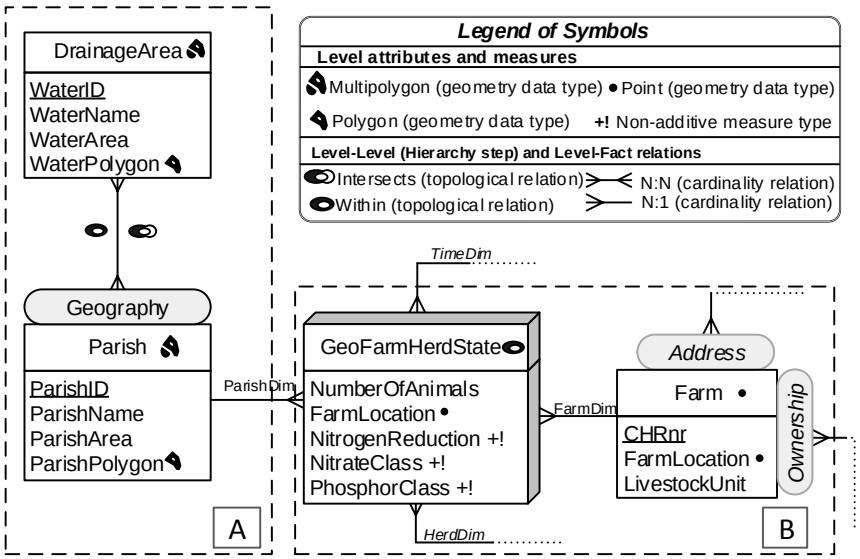


Fig. 22: GeoFarmHerdState (spatial) MD schema of livestock holdings data (Adapted from [14])

Parish, Farm, and Drainage Area instances from the use case are depicted on a map (Figure 21) and explained in Example 17 in Section 5. A hierarchy example from the instances given in the map figure is depicted as a graph in Figure 23. The graph shows the topological relations between level members (parish-drainage area) and between fact members and level members (farms-

parish). SOLAP operators aggregate measures (from farm states) at different granularity levels of the spatial hierarchy. In order to aggregate the measures correctly during a s-roll-up operation, the topological relation between the level members should be taken into consideration, where two relations are available: *within* and *intersects* (black and red arrows in Figure 23).

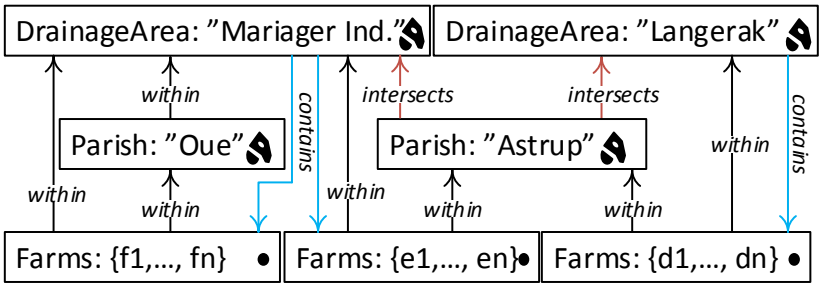


Fig. 23: Spatial Hierarchy Example for SOLAP (Adapted from [14])

For example, if we would like to aggregate the total number of animals from parishes to drainage areas, there is a great chance that we aggregate the measure values incorrectly from Astrup (parish) to drainage areas (Mariager Inderfjord and Langerak) by counting twice, since parish Astrup *intersects* with both of the drainage areas. Due to the polygon geometry of the level members, roll-up relation encounters a *many-to-many* (cardinality) relationship when a parish *intersects* with a drainage area. To prevent falsely aggregating the measure values, a SOLAP operation with a spatial drill-down from drainage area members to fact members, using the *contains* topological relation is required (blue arrows in Figure 23). Another way to prevent this issue can be achieved by de-normalizing the fact table with redundant links from farm (fact) members to drainage area (level) members, where we create those links explicitly with direct *within* relations.

QB4SOLAP vocabulary allows us to annotate such topological relations between level members and between fact members and level members with well-defined predicates, where QB4OLAP only supports `skos:broader` predicate between level members of a hierarchy and referential integrity keys between fact members and level members.

In RDF2SOLAP spatial enrichment, we consider two approaches: 1) Hierarchical enrichment: We enrich the annotation of hierarchy steps between level members with direct topological relations from QB4SOLAP vocabulary instead of using `skos:broader` predicate. 2) Factual enrichment: We denormalize the logical arrangement of spatial levels by directly annotating the topological relations from fact members to a higher parent level member.

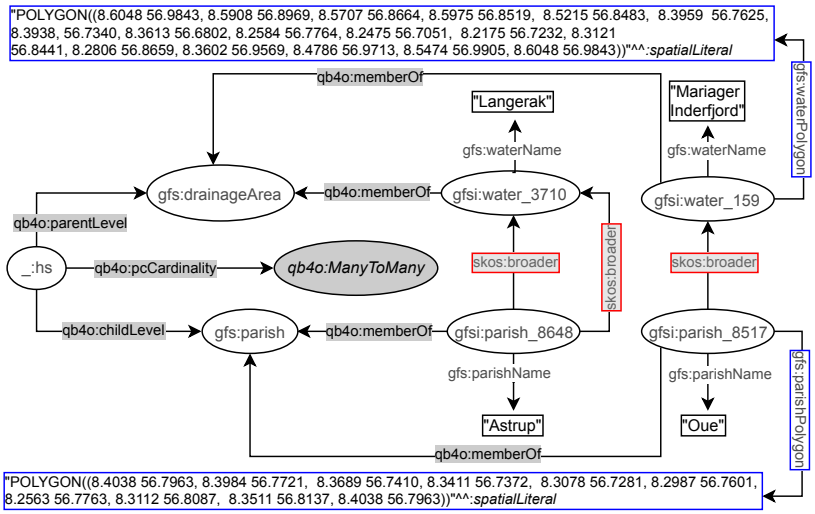


Fig. 24: Hierarchy steps in QB4OLAP before MD enrichment(Adapted from [14])

In Figure 24, QB4OLAP annotation of the hierarchy step from `gfs:parish` level to `gfs:drainageArea` level is given. We use similar conventions to prefix the schema members (level, fact, etc.) with `gfs:` and instance members (level members, fact members) with `gfsi:` as explained in Section 5. The relations between the level members are given with `skos:broader` predicate, highlighted in red boxes. The geometry attribute values are highlighted in blue boxes, which is an indicator that we can enrich the hierarchy steps between the level members with well-defined topological relations by utilizing these geometry attributes (in spatial Boolean functions).

Figure 25 depicts the QB4SOLAP annotation of the hierarchy step from Fig. 24 after the enrichment with topological relations (highlighted in green boxes) from the QB4SOLAP vocabulary. Note that, the enrichment process also re-defines the fact schema, where we enrich the data structure definition (DSD), with topological relations that are found between the level members that can be seen on the left part of the figure with the elements highlighted with a green border.

6.3 RDF2SOLAP Enrichment Algorithms

Spatial enrichment in RDF2SOLAP occurs in two phases, as briefly mentioned in our approach above (Section 6.2). The first one is the hierarchical enrichment phase and the second one is the factual enrichment phase. Before giving details on the enrichment phases and algorithms within, spatial helper functions are briefly explained below.

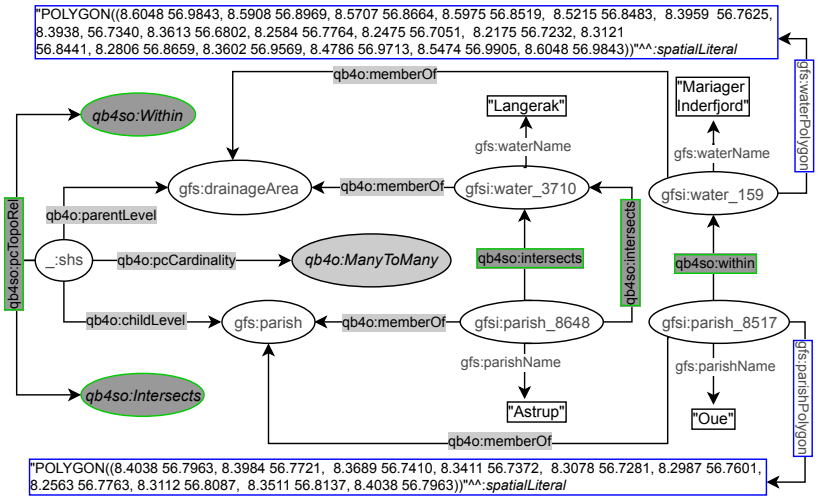


Fig. 25: Spatial hierarchy steps in QB4SOLAP after MD enrichment(Adapted from [14])

Spatial Helper Functions

Spatial helper functions are utilized in both enrichment phases to identify the spatial values of the geometries from the input data and use those values to find a topological relation as an output. We have implemented two spatial helper functions.

The first one is $\texttt{getSpatialValues}(\mathcal{G}^I_{A(lm)})$:$V_{s(a)}$, which takes an RDF graph of attributes of level members (or can be fact members) as an input, and simply scans the input data to find spatial geometry values and returns a set of spatial values that are filtered in the attributes of level members (or measure values of fact members). These spatial values are coordinates describing the geographical location and shape of the level member or fact member as an attribute or measure value correspondingly.

The second one is: $\texttt{relateSpatialValues}(v_{a_c}, v_{a_p})$:$\texttt{topoRel}_i$, which takes a pair of spatial values (from child and parent level members) and returns the topological relation between those values. In order to design and implement the second helper function we have used Table 10. The table contains topological relations from DE-9DIM - Dimensionally Extended Nine-Intersection [9] model, which describes the possible Boolean relations of two geometries in two-dimensional space.

We focus on three main simple geometry types: *point*, *line*, and *polygon*, which can represent spatial attribute values of level members or spatial measure values of fact members. In a roll-up relation[39], hierarchically and topo-

[39] A roll-up relation can occur from child level to parent level between level members, or from fact to (base) level between fact members and level members.

Table 10: Topological relations (✓: hierarchically and topologically applicable, ×: topologically not applicable, –: hierarchically not applicable) - (Adapted from [14])

Roll-up Relations	child level	point (pt.)			line (ln.)			polygon (po.)		
	parent level	pt.	ln.	po.	pt.	ln.	po.	pt.	ln.	po.
Topological Relations	within	×	✓	✓	–	✓	✓	–	–	✓
	contains	–	–	–	–	–	–	–	–	–
	intersects	✓	✓	✓	–	✓	✓	–	–	✓
	touches	×	×	×	–	✓	✓	–	–	✓
	overlaps	×	×	×	–	✓	✓	–	–	✓
	crosses	×	×	×	–	✓	✓	–	–	×
	coveredBy	×	×	×	–	×	✓	–	–	✓
	covers	–	–	–	–	–	–	–	–	–
	equals	✓	×	×	–	✓	×	–	–	✓

logically applicable relations are given with a checkmark (✓) sign. Top-down topological relations such as *contains* and *covers* are considered as hierarchically not applicable, since a lower level member (i.e., child-level) cannot contain or cover a higher-level member (i.e., parent level), therefore the entire row with contains and covers topological relations are given with dash (–) sign. Another example case of hierarchically not applicable relations occurs in between *line-point*, *polygon-point*, and *polygon-line* geometries, since a parent level member needs to have geometry type of the same or higher dimensionality of the child level member, where point geometry is 0-dimensional, a line is 1-dimensional and a polygon is 2-dimensional, etc. [14]. Topologically not applicable relations are inherited from DE-9DIM and marked with (×) cross sign accordingly.

Figure 26 depicts the topologically and hierarchically applicable relations with geometry pairs from Table 10. Topological relations are simplified in the figure, where we generalize some of the relations when possible. For example if a line *crosses* a polygon at two points or a line *touches* a polygon at a single point, both cases are generalized as a line *intersects* polygon (Figure 26(e)). These generalizations are used to design and implement the algorithm `relateSpatialValues`(v_{a_c}, v_{a_p}):`topoRel`$_i$.

Hierarchical Enrichment

We create hierarchical enrichment algorithms by exploiting the existing non-spatial QB4OLAP semantics. We distinguish two cases in our algorithms that are explained along with the algorithms (Alg. 4 and Alg. 5).

The first case is to find *explicit* (spatial) hierarchy steps, where it is perceived that there are direct roll-up relations between the level members that are annotated with `skos:broader` predicate (as shown in Figure 24). So we

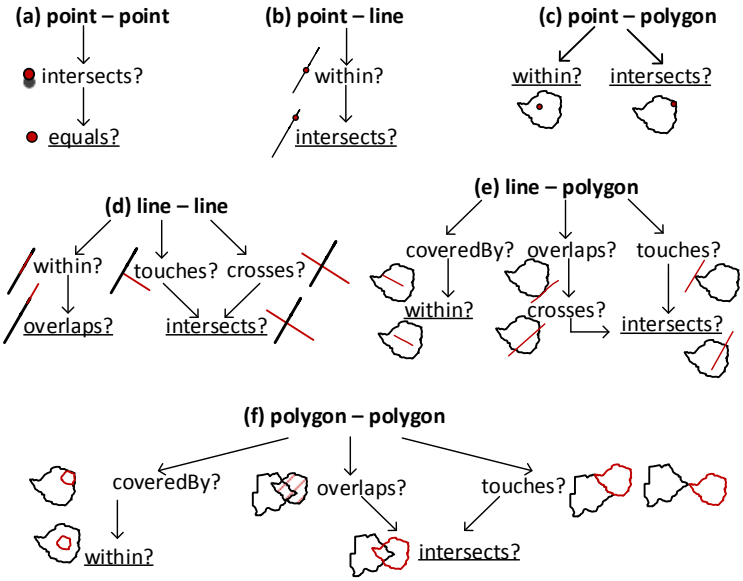


Fig. 26: Simplifying topological relations (Adapted from [14])

can easily detect these explicit hierarchy steps, and use their geometry attributes if there is any to find the topological relations between those level members. The algorithm (Alg. 4) `detectSpatialHS`$(\mathcal{G}^I_{RU(hs)}, \mathcal{G}^I_{A(lm)}) : \mathcal{G}^I_{RU(shs)}$ takes the explicit roll-up relations of hierarchy steps ($\mathcal{G}^I_{RU(hs)}$) and attributes of level members ($\mathcal{G}^I_{A(lm)}$) and returns the *detected* spatial hierarchy steps ($\mathcal{G}^I_{RU(shs)}$) as a result.

In Line 4 of the algorithm in the **foreach** loop, explicit `skos:broader` relations are filtered and attribute values of the level members (for parent-child levels) are retrieved. These attribute values (of level members) are given as an input (in Lines 6 and 9) to the first helper function `getSpatialValues` to find the geometry attributes. If there are geometry attributes for both child-level members, pairs of these spatial values are given as an input to the second helper function `releateSpatialValues` (Line 12). The result RDF graph of roll-up relations with detected spatial hierarchy steps ($\mathcal{G}^I_{RU(shs)}$) is incrementally added with the triple patterns of the identifier of the child level member, topological relation, and the identifier of the parent level member (Line 14) and returned as the result in Line 15.

The second case is to find *implicit* (spatial) hierarchy steps between the level members, where there are not any direct roll-up relations that are annotated between the level members. In order to handle this situation, where there

Algorithm 4: detectSpatialHS($\mathcal{G}^I_{RU(hs)}, \mathcal{G}^I_{A(lm)}$) : $\mathcal{G}^I_{RU(shs)}$ (Adapted from [14])

Input: $\mathcal{G}^I_{A(lm)}$, $\mathcal{G}^I_{RU(hs)}$
Output: $\mathcal{G}^I_{RU(shs)}$

begin
  $\mathcal{G}^I_{RU(shs)} = \emptyset$; /*initialize output graph as emptyset*/
  $\mathcal{G}^I_{A(lm_c)} = \emptyset$; $\mathcal{G}^I_{A(lm_p)} = \emptyset$; $V_{s(a_c)} = \emptyset$; $V_{s(a_p)} = \emptyset$; $\texttt{topoRel}_i = null$; /*temporary variable and sets*/
  foreach $((id^I(lm_c)\ id^S(a_c)\ v_{a_c}), (id^I(lm_p)\ id^S(a_p)\ v_{a_p})) \mid (id^I(lm_c)\ id^S(a_c)\ v_{a_c}), (id^I(lm_p)\ id^S(a_p)\ v_{a_p}) \in \mathcal{G}^I_{A(lm)} \wedge (id^I(lm_c)\ \texttt{skos:broader}\ id^I(lm_p)) \in \mathcal{G}^I_{RU(hs)} \wedge lm_c \rightsquigarrow v_{a_c} \wedge\ lm_p \rightsquigarrow v_{a_p} \wedge lm_c \sqsubseteq lm_p$ **do**
    $\mathcal{G}^I_{A(lm_c)} = \{(id^I(lm_c)\ id^S(a_c)\ v_{a_c})\}$;
    $V_{s(a_c)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_c)})$;
    if $V_{s(a_c)} \neq \emptyset$ **then**
      $\mathcal{G}^I_{A(lm_p)} = \{(id^I(lm_p)\ id^S(a_p)\ v_{a_p})\}$;
      $V_{s(a_p)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_p)})$;
      if $V_{s(a_p)} \neq \emptyset$ **then**
        foreach $(v_{a_c}, v_{a_p}) \in V_{s(a_c)} \times V_{s(a_p)}$ **do**
          $\texttt{topoRel}_i = \texttt{relateSpatialValues}(v_{a_c}, v_{a_p})$;
          if $\texttt{topoRel}_i \neq null$ **then**
            $\mathcal{G}^I_{RU(shs)} \cup = \{(id^I(lm_c)\ \texttt{topoRel}_i\ id^I(lm_p))\}$;
  return $\mathcal{G}^I_{RU(shs)}$

are not any explicit hierarchy steps between the level members, we benefit from QB4OLAP schema graphs for dimensions, hierarchies, and levels. This way we can classify dimensions and hierarchies and make sure that we are extracting the level member pairs for the levels in the same hierarchy of the same dimension and comparing only the attribute values of these level members to find the topological relations of the spatial hierarchy steps. The algorithm (Alg. 5) $\texttt{discoverSpatialHS}(\mathcal{G}^S_D, \mathcal{G}^S_{H(d)}, \mathcal{G}^S_{L(h)}, \mathcal{G}^I_{LM(l)}, \mathcal{G}^I_{A(lm)})$: $\mathcal{G}^I_{RU(shs)}$ takes the dimensions ($\mathcal{G}^S_D$) and hierarchies in the dimension ($\mathcal{G}^S_{H(d)}$) and levels in the hierarchy ($\mathcal{G}^S_{L(h)}$) from the schema graphs, and and level members of the level ($\mathcal{G}^I_{LM(l)}$) and attributes of the level members ($\mathcal{G}^I_{A(lm)}$) from the instance graphs as inputs and returns the *discovered* spatial hierarchy steps

($\mathcal{G}^{I}_{RU(shs)}$) as a result.

In each **foreach** loop (in Lines 5, 6, and 7) of schema elements we iterate through the dimensions, hierarchies, and levels in the hierarchy, to create level pairs (in Line 8) and get level members of these level pairs by filtering with `qb4o:memberOf` predicate (in Line 9). The algorithm iterates through the level member pairs and their attributes of the level members, where we can follow a similar logic to filter the spatial attribute values from the level members' attributes as in the previous algorithm by using `getSpatialValues` helper function (in Line 13). If there are geometry attributes for both of the level members (Line 14), pairs of these spatial values are given as an input to the second helper function `releateSpatialValues` (Lines 15 and 16). The result RDF graph of roll-up relations with discovered spatial hierarchy steps ($\mathcal{G}^{I}_{RU(shs)}$) is incrementally added with the triple patterns of the identifier of the level member n, topological relation, and the identifier of level member k (Line 18) and returned as the result in Line 19.

Factual Enrichment

In principle, the factual enrichment phase is very similar to the hierarchical enrichment phase, although, the roll-up path includes not only the roll-up relations (hierarchy steps) between level members but also the roll-up relations between fact members and level members. In a multi-dimensional fact schema, a roll-up path between fact members and level members are defined through explicit (direct) relations with referential integrity keys between a fact member and the base level member of the dimensions. Through the roll-up paths between fact members and the base level member of the dimension, we can derive new perspectives by aggregating measures in SOLAP operations. In spatial data warehouses, facts usually have spatial measures, where we can detect topological relations through the explicit roll-up paths between fact members and base level members of the dimension. QB4SOLAP allows users to represent fact-level topological relations both at the schema level (in the DSD) and the instance level. Examples of a (spatial) fact schema (DSD) and fact members are given in Section 5 in Examples 15 and 16. In the following, we recap the examples with RDF2SOLAP spatial enrichment with the highlighted lines before explaining the factual enrichment algorithms.

Example 20 (Fact Schema and Fact Members (Adapted from [14]))
Factual enrichment at the schema level occurs when the DSD is re-defined with spatial concepts. QB4SOLAP represents the topological relations between fact and base level members in addition to the *cardinality* relations. For example, the cardinality relation between fact farm-state and level parish is many-to-one (Line 4), and the topological relations are defined as an instance of `qb4so:Within` topological relation class (Line 5). This

Algorithm 5: discoverSpatialHS($\mathcal{G}^S_D, \mathcal{G}^S_{H(d)}, \mathcal{G}^S_{L(h)}, \mathcal{G}^I_{LM(l)}, \mathcal{G}^I_{A(lm)}$): $\mathcal{G}^I_{RU(shs)}$ (Adapted from [14])

Input: $\mathcal{G}^S_D$, $\mathcal{G}^S_{H(d)}$, $\mathcal{G}^S_{L(h)}$, $\mathcal{G}^I_{LM(l)}$, $\mathcal{G}^I_{A(lm)}$

Output: $\mathcal{G}^I_{RU(shs)}$

begin

  $\mathcal{G}^I_{RU(shs)} = \emptyset$; $\texttt{topoRel}_i = null$ /*initialize the output graph as an empty set and a temporary variable as null*/

  $V_{s(a_n)} = \emptyset$; $V_{s(a_k)} = \emptyset$; /*initialize temporary sets as empty sets for keeping spatial attribute values*/

  $\mathcal{G}^I_{A(lm_n)} = \emptyset$; $\mathcal{G}^I_{A(lm_k)} = \emptyset$; /*initialize empty sets to keep triple patterns for attributes of level members*/

  foreach $(id^S(d)$ `qb4o:hasHierarchy` $id^S(h)) \in \mathcal{G}^S_D$ /*iterate through the dimensions*/ **do**

    foreach $(id^S(h)$ `qb4o:inDimension` $id^S(d)) \in \mathcal{G}^S_H(d)$ /*iterate through the hierarchies*/ **do**

      foreach $(id^S(h)$ `qb4o:hasLevel` $id^S(l)) \in \mathcal{G}^S_H(d)$ /*while iterating through the levels in the hiearchy, get level pairs next*/ **do**

        foreach $(id^S(l_i), id^S(l_j)) \in \mathcal{G}^S_{L(h)} \times \mathcal{G}^S_{L(h)} \mid id^S(l_i) \neq id^S(l_j) \wedge$

        $\bigcup_{lm \in LM(l)}((id^I(lm)$ `qb4o:memberOf` $id^S(l_i)), (id^I(lm)$ `qb4o:memberOf` $id^S(l_j))) \in \mathcal{G}^I_{LM(l)}$ /*in each level pair, while iterating through their level members, get a pair of level members $(id^I(lm_n), id^I(lm_k)$, where each level member comes from different levels*/ **do**

          foreach $(id^I(lm_n), id^I(lm_k)) \in \mathcal{G}^I_{LM(l)} \times \mathcal{G}^I_{LM(l)} \mid id^I(lm_n) \neq id^I(lm_k) \wedge id^I(lm_n) \in \mathcal{G}^I_{LM(l_i)} \implies id^I(lm_k) \in \mathcal{G}^I_{LM(l_j)} \mid \mathcal{G}^I_{LM(l_i)} \subset \mathcal{G}^I_{LM(l)} \wedge \mathcal{G}^I_{LM(l_j)} \subset \mathcal{G}^I_{LM(l)} \wedge \mathcal{G}^I_{LM(l_i)} \neq \mathcal{G}^I_{LM(l_j)}$ /*iterate through the pairs of level members*/ **do**

            foreach $((id^I(lm_n)\ id^S(a_i)\ v_{a_i}), ((id^I(lm_k)\ id^S(a_j)\ v_{a_j})) \in \mathcal{G}^I_{A(lm)} \times \mathcal{G}^I_{A(lm)}$ /*iterate through the pairs of level members' attributes*/ **do**

              $\mathcal{G}^I_{A(lm_n)} = \{(id^I(lm_n)\ id^S(a_i)\ v_{a_i})\}$; $\mathcal{G}^I_{A(lm_k)} = \{(id^I(lm_k)\ id^S(a_j)\ v_{a_j})\}$;

              $V_{s(a_n)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_n)})$;

              $V_{s(a_k)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_k)})$;

              if $V_{s(a_n)} \neq \emptyset \wedge V_{s(a_k)} \neq \emptyset$ /*make sure there are spatial values in the temporary sets*/ **then**

                foreach $(v_{a_i}, v_{a_j}) \in V_{s(a_n)} \times V_{s(a_k)}$ **do**

                  $\texttt{topoRel}_i = \texttt{relateSpatialValues}(v_{a_i}, v_{a_j})$;

                  if $\texttt{topoRel}_i \neq null$ /*make sure there is a topological relation assigned to the variable*/ **then**

                    $\mathcal{G}^I_{RU(shs)} \cup = \{(id^I(lm_n)\ \texttt{topoRel}_i\ id^I(lm_k))\}$;

  return $\mathcal{G}^I_{RU(shs)}$

relation can be derived only if the fact members (at the instance level) are enriched and annotated with explicit topological relations as given in Line 12.

QB4SOLAP also extends the fact schema with spatial aggregate functions that are defined over spatial measures (if there are any). An example of aggregate function for farm location spatial measure (with point geometry) can be defined as an instance of `qb4so:ConvexHull` class (Line 7).

```
##Spatial Fact Schema in QB4SOLAP##
1 gfs:GeoFarmHerdState a qb:DataStructureDefinition ;
#Lowest spatial level for each dimension in the cube#
2 qb:component [qb4o:level gfs:farm ; qb4o:cardinality qb4o:ManyToOne ;
3     qb4so:topologicalRelation qb4so:Equals] ;
4 qb:component [qb4o:level gfs:parish ; qb4o:cardinality qb4o:ManyToOne ;
5     qb4so:topologicalRelation qb4so:Within] ;
 #Example of a spatial measure in the cube#
6 qb:component [qb:measure gfs:farmLocation ;
7     qb4o:aggregateFunction qb4so:ConvexHull] .
```

Factual enrichment at the instance level between fact members and level members is exemplified in Line 12 for explicit relations (because there is a direct reference in Line 9) and in Line 13 for implicit relations.

```
 ##GeoFarmHerdState cube:  observation fact example##
8 gfsi:farmState_103850_12_2015 a qb:Observation ;
9    gfs:farm gfsi:farm_103850 ; gfs:parish gfsi:parish_8648 ;
10   gfs:livestockUnit "4.2699999999999996"^^xsd:double ;
11   gfs:farmLocation "POINT (8.31941 56.75822)"^^geo:spatialLiteral ;
12   qb4so:equals gfsi:farm_103850 ; qb4so:within gfsi:parish_8648 ;
13   qb4so:within gfsi:water_3770 .
```

Factual enrichment at the instance level has two similar cases as in hierarchical enrichment.

The first case is to find *explicit* fact-level relations, where there is a direct link between the fact member and base level member via referential integrity keys. Alg. 5 from Paper F - `detectFactLevelRelations`$(\mathcal{G}^I_{FM(F)}, \mathcal{G}^I_{A(lm)})$: $\mathcal{G}^I_{FM(F_s)}$ uses a similar approach as in Alg. 4 to detect hierarchy steps between the level members. Instead of the explicit roll-up relation between parent-child level members, we get the fact members where there is a direct relation to the base level members from the fact member as given in the Listing above in Ex. 20 (Line 9). By using this explicit relation we get pairs of fact-base level members and detect the topological relations between the spatial measure of

the fact member and spatial attribute of the level members, which is added to the existing fact members and returned as a result (Line 12).

The second case is to find *implict* fact-level relations, where there is not any direct link between the fact member and base level member via referential integrity keys or to find an *implicit* topological relation between fact members and higher granularity (parent) level members. The algorithm (Alg. 6 from Paper F) `discoverFactLevelRelations`($\mathcal{G}^I_{FM(F)}$, $\mathcal{G}^I_{LM(l)}$, $\mathcal{G}^I_{A(lm)}$, $\mathcal{G}^S_D$, $\mathcal{G}^S_{H(d)}$, $\mathcal{G}^S_{HS(h)}$) : $\mathcal{G}^I_{FM(F_s)}$ uses an approach similar to discovering implicit hierarchy steps (Alg. 5) by utilizing QB4OLAP schema definitions of multi-dimensional concepts such as dimensions ($\mathcal{G}^S_D$), hierarchies in the dimension ($\mathcal{G}^S_{H(d)}$) and hierarchy steps in the hierarchy ($\mathcal{G}^S_{HS(h)}$). This way we can identify all the base level members[40] and find the missing roll-up relations between the fact member and level members, and annotate those with topological relations (Line 12). The other case that we would like to establish a direct link with topological relations between a fact member and level member, which is not a base level is when there are many-to-many relations between the levels in a hierarchy. As explained in Section 6.2 along with Fig. 23, many-to-many relations between levels we encounter the problem of aggregating measures incorrectly. Therefore, fact-level de-normalization by creating redundant links that define the direct relation between level members and fact members can remedy the problem. By using the QB4OLAP annotations for hierarchy steps we can as well reveal the levels that have many-to-many relations[41]. After identifying the level members at a higher granularity (drainage area level in this case) that can benefit from direct links to fact members, we use the same approach to discover topological relations (as in the previous algorithms) and enrich the fact members by annotating with the *discovered* topological relations (Line 13 in Ex. 20).

Finally, factual enrichment at the schema level occurs by re-defining the fact schema using the results from the factual enrichment at the instance level. The algorithm (Alg. 6) `defineSpatialFactDSD`($\mathcal{G}^I_{FM(F_s)}$, $\mathcal{G}^S_F$) : $\mathcal{G}^S_{F_s}$ takes the fact schema ($\mathcal{G}^S_F$) and spatially enriched fact members ($\mathcal{G}^I_{FM(F_s)}$) as a result of `detectFactLevelRelations` and `discoverFactLevelRelations` –the previous two algorithms– and returns the new fact schema ($\mathcal{G}^S_{F_s}$) that is spatially enriched.

An example of the (non-spatial) fact schema is given in Ex. 20 in the listing lines 1-7 (please ignore the blue lines 3, 5, and 7 for now). In the first **foreach** statement in Line 3 (Alg. 6) we get the spatially enriched fact members and for every topological relation from the next **foreach** loop (Line 4), we annotate

[40] In QB4OLAP hierarchy steps, levels are annotated as parent level or child level (see Ex. 3), if a level has never been annotated as a parent level, thus it is a base level and all the level members in this level are base-level members.

[41] In QB4OLAP hierarchy steps, levels are annotated as well with cardinality relations (Ex. 3).

them in the fact schema (Line 5). Thus, the example Listing (Ex. 20) is added with the blue lines 3 and 5. Next **foreach** statement for the fact members (Line 6) finds the spatial measures and based on the geometry data type of the spatial measures, an aggregate function[42] is suggested to be added to the enriched fact schema. As a result of this, the example Listing (Ex. 20) is added with the blue annotation in line 7, showing that the aggregate function over the spatial measure (farm location), which has a point geometry (Ex.20, Line 11 in the instance data) is suggested as convex hull.

6.4 Implementation Details

In order to implement our approach (presented with the enrichment algorithms), on top of triple stores using the semantic web technologies, we have to make some implementation decisions. We stored the QB4OLAP (instance and schema) data in native RDF format in a triple store (*Virtuoso version 07.20.3217*). The triple store we chose to store RDF data (Virtuoso Open Source) supports many geometry data types (i.e., POLYGON, MULTIPOLYGON), although, it does not support all the spatial Boolean functions from DE9DIM model given in Table 10. The open-source version of Virtuoso only supports *intersects*, *contains*, and *within* with built-in database functions. Therefore, We decided to implement RDF2SOLAP independently from the spatial capabilities of the triple stores, by using a third-party web tool for spatial analysis. We implemented RDF2SOLAP using the Node.js platform, and as the third party web tool, we have used a Javascript library that is called *Turfjs*. This way, we can ensure RDF2SOLAP can be used on top of any triple store since the Javascript library provides us the spatial analysis capabilities and a flexible development environment, independent from any choice of the triple store.

The details of the implementation, SPARQL endpoints, and RDF data sets can be found on our project page[43]. The code repository for the whole implementation can be found on GitHub[44].

6.5 Implementation Results

RDF2SOLAP implementation results for hierarchical enrichment algorithms (`detectSpatialHS` and `discoverSpatialHS`) and for factual enrichment algorithms (`detectFactLevelRelations` and `discoverFactLevelRelations`) over the instance data are summarized in Table 11. For each algorithm, we have

[42] In QB4OLAP, `qb4o:AggregateFunction` class has only instances (e.g., `qb4o:Avg`, `qb4o:Sum` functions) for numerical measures. QB4SOLAP extends this class with a subclass `qb4so:SpatialAggregateFunction`, which has instances of spatial aggregate functions (e.g., `qb4so:ConvexHull`, `qb4so:Union`) for spatial measures [12, 15] (Adapted from [14]).

[43] Project Page: http://extbi.cs.aau.dk/RDF2SOLAP

[44] RDF2SOLAP Repository: https://github.com/lopno/rdf2solap

Algorithm 6: defineSpatialFactDSD($\mathcal{G}^I_{FM(F_s)}, \mathcal{G}^S_F$) : $\mathcal{G}^S_{F_s}$
(Adapted from [14])

Input: $\mathcal{G}^I_{FM(F_s)}, \mathcal{G}^S_F$
Output: $\mathcal{G}^S_{F_s}$

begin
  $\mathcal{G}^S_{F_s} = \mathcal{G}^S_F$; $\texttt{aggFunc}_i = null$; /*initalize the output graph and temporary variable*/
  foreach $(id^I(f_i)$ `rdf:type qb:Observation`$) \in \mathcal{G}^I_{FM_{(F_s)}}$ **do**
    foreach $(id^I(f_i)$ $\texttt{topoRel}_i$ $id^I(lm_j)) \in \mathcal{G}^I_{FM_{(F_s)}}$ | $\bigcup_{l_n \in L(f_i)} (id^I(f_i)\ id^S(l_n)\ id^I(lm_j)) \in \mathcal{G}^I_{FM_{(F_s)}}$ /*each $\texttt{topoRel}_i$ in the fact member triples goes into the DSD with its corresponding level l_n*/ **do**
      $\mathcal{G}^S_{F_{(F_s)}} \cup = \{(id^S(F)$ `qb:component [qb4o:level` $id^S(l_n)$, `qb4so:topologicalRelation` $id^S(\texttt{topoRel}_i)$`]`$)\}$;
    foreach $v_{m_k} \in (id^I(f_i)\ id^S(m_k)\ v_{m_k})$ /*find the spatial measures from the fact triples*/ **do**
      if v_{m_k} *is a* `geo:spatialLiteral` **then**
        switch *(geoType*(v_{m_k})*)* /**geoType*(v_a) function returns the geometry type of a given attribute value*/ **do**
          case (`POINT`) /*point geometry measures are supported to be aggregated with ConvexHull function*/ **do**
            $\texttt{aggFunc}_i =$ `qb4so:ConvexHull`
          case (`LINE`) /*line geometry measures are supported to be aggregated with Union function*/ **do**
            $\texttt{aggFunc}_i =$ `qb4so:Union`
          case (`POLYGON`) /*polygon geometry measures are supported to be aggregated with Union, Centroid,*/ **do**
            $\texttt{aggFunc}_i =$ `qb4so:Union` $\vee$ `qb4so:Centroid` $\vee$ `qb4so:MBR` /*or MBR functions*/
        $\mathcal{G}^S_{F_{(F_s)}} \cup = \{(id^S(F)$ `qb:component [qb:measure` $id^S(m_k)$, `qb4o:aggregateFunction` $id^S(\texttt{aggFunc}_i)$`]`$)\}$;
  return $\mathcal{G}^S_{F_s}$

Table 11: Implementation Results (f.s.= farm states, p.= parishes, d.a.= drainage areas) - Adapted from [14]

	INPUT			OUTPUT		
	Child Members#	Parent Members#	Explicit Relations#	TopoRel#		Run times
`detectSpatialHS`	p.: 2,180	d.a.: 134	2,683	intersects	636	29 s
				within	2,046	
`detectFactLevelR.`	f.s.: 40,039	p.: 2,180	39,800	within	39,334	7 s
`discoverSpatialHS`	p.: 2,180	d.a.: 134	NONE	intersects	1,088	2,622 s
				within	3,392	
`discoverFactLevelR.`	f.s.: 40,039	p.: 2,180	NONE	within	39,998	1,920 s
	f.s.: 40,039	d.a.: 134	NONE	within	39,845	525 s

created test cases in RDF2SOLAP and queried the SPARQL endpoint of the triple store, where we extracted the data in JSON format to Node.js.

As depicted in Figure 23, parishes, and drainage areas have a multi-polygon data type. From a practical point of view, we have kept this multi-polygon data in triple store as multi-part polygons, which implies that several parishes or drainage areas with the same unique URI have different polygon coordinates. When parishes or drainage areas are grouped by their unique IDs (URIs), the different polygons can compose and represent the multi-polygon geometry of the parish or the drainage area. We have implemented a *bounding box* function on the multi-part polygon data, where we group them by their unique URI and put in a bounding box, thus we can call the spatial Boolean functions (i.e, *within*, *intersects*) between parish and drainage area members in hierarchical enrichment algorithms.

The first two algorithms (prefixed with `detect...`) in Table 11 present the results, for detecting explicit relations, where a direct relation between the level members (via `skos:broader` predicate) or between the fact members and base level members (via referential integrity keys) are provided as an INPUT as well. The OUTPUT column presents the number of topological relations found and provides the run times (in seconds). The last two algorithms (prefixed with `discover...`) present the results, for discovering implicit relations, where there are NONE direct (explicit) relations provided as an INPUT, between child members and parent members. The input datasets are summarized as 2,180 parish members, 134 drainage area members, and 40,039 farm-state members. Between parish members (polygon) and drainage area members (polygon), 2,683 explicit relations are provided, where 39,800 explicit relations are provided between the farm-state members (point) and parish members (polygon).

Next, we briefly mention the algorithm run-times. The evaluation of the overall performance of RDF2SOLAP enrichment is given in the Quantitative Evaluation in Section 6.6. By looking at the results in the table, we can notice that the most expensive algorithm is `discoverSpatialHS` with run time 2,622

seconds. Since, there are no explicit relations are provided the algorithm calls the spatial Boolean functions $134 \times 2,180 = 29,2120$ times for each function. Note that, we also group the unique URIs of the polygon coordinates to create bounding boxes, where, in this algorithm, both of the input data sets (parish-drainage area) have polygon geometries. On the other hand, the algorithm `detectSpatialHS` with explicit relations results in *29* seconds, since the algorithm utilizes the given relations and checks for the spatial Boolean functions only $2,683$ times. The algorithm `detectFactLevelRelations` (between fact level members/farm states and base level members/parishes) performs the fastest - resulting in *7* seconds. Out of $39,800$ explicit relations provided between fact states and parishes, $39,334$ topological relations are detected as a result of the algorithm. On the other hand, the algorithm `discoverFactLevelRelations` performs slower as expected ($1,920$ seconds), and discovers more topological relations between farm states and parishes, where there might be missing links that were not provided with explicit relations.

6.6 Evaluation and Discussion

For evaluating the performance of our approach, we measure and present the total time to get similar results from the RDF data in two different non-SW environments. The results are presented under quantitative evaluation. We also compare the RDF2SOLAP enrichment results in terms of accuracy (number of topological relations found) and coverage in the qualitative evaluation subsection against these two different environments. Finally, we discuss the technical lessons and summarize our work.

Quantitative Evaluation

We compare the query run times of the algorithms in RDF2SOLAP with two different query platforms: an RDBMS tool and a GIS tool. Since these tools cannot process RDF data natively, we consider the data preparation and load times as development costs (Table 12). Development cost is given in hours, which involves, extracting the data, loading into these (non-SW) environments in their native format. We assume that the developer has a basic knowledge of the domain, data set, the schema of the data set, and able to extract data with SPARQL queries, can perform SQL queries on RDBMS to get similar results as in the algorithms, and also knows how to operate through the interface of the GIS tool. The development cost excludes the preparation (downloading and installing) of the environments. The development cost of RDF2SOLAP is a configuration set-up, where the user should point to the SPARQL endpoint where the instance triples are located and specify the RDF cube schema of the use case data.

Table 12: Performance Evaluation Results (f.s.= farm states, p.= parishes, d.a.= drainage areas) - Adapted from [14]

	Query Platform	Performance Results	
		Run times	Development cost
`detectSpatialHS` (p.– d.a.)	RDF2SOLAP	29 s	5 min.
	RDBMS	< 1 s	1-1.5 days
`detectFactLevelR.` (f.s. – p.)	RDF2SOLAP	7 s	5 min.
	RDBMS	< 1 s	1-1.5 days
`discoverSpatialHS` (p. – d.a.)	RDF2SOLAP	2,622	5 min.
	RDBMS	43 s	1-1.5 days
	GIS	45 s	2 days
`discoverFactLevelR.` (f.s. – p.)	RDF2SOLAP	1,920 s	5 min.
	RDBMS	95 s	1-1.5 days
	GIS	72 s	2 days
`discoverFactLevelR.` (f.s. - d.a.)	RDF2SOLAP	525 s	5 min.
	RDBMS	48 s	1-1.5 days
	GIS	41 s	2 days

In RDBMS, after the corresponding data sets are extracted from the triple store and loaded into RDBMS in a relational format, and queries are prepared for getting similar results as in algorithms, the query run times for detecting explicit relations are less than 1 second. However, the preparation steps take around 1-1.5 working days, while in RDF2SOLAP configuration is prepared natively within 5 minutes for fetching the data sets from the endpoint. Note that, the GIS tool is not involved in detecting explicit relation algorithms, since, the tool employs spatial joins instead of joining through referential integrity of explicit relations.

In the GIS tool, preparation times are a little bit longer than the RDBMS, where we have converted the relational data into shape format and prepared layers from the corresponding data sets to perform spatial joins. In total, we spent around 2 working days to make the data ready for query processing. When the data is ready, the GIS tool mostly outperformed in discovering topological relations.

Query run times in RDF2SOLAP involve, parsing the RDF data in JSON, calling the helper functions, returning bounding box objects for multi-part polygon data. Therefore, RDF2SOLAP demonstrated adequate performance at a very low development cost compared to the non-SW query platforms, where the configuration of the RDF2SOLAP enrichment process can be done within 5 minutes. The configuration requires only to point to the SPARQL endpoint, where the RDF cube schema namespace URI is located, then, the test cases for running the RDF2SOLAP enrichment process are created by receiving the input parameters to the enrichment algorithms. RDF2SOLAP provides a significant contribution to the users by cutting down the devel-

opment costs on data extraction and preparation times by 2 to 3 orders of magnitude, where 1.5-2 days of development cost is reduced to 5 minutes. Besides RDF2SOLAP can natively operate over a wide range of SPARQL endpoints, where the choice of a triple store at the back-end of the endpoint does not have any implications for the enrichment process as we use a third-party JavaScript library as a wrapper for the enrichment algorithms.

Qualitative Evaluation

In qualitative evaluation, we compare the number of topological relations found as a result of running the algorithms and queries in three different platforms: GIS, RDBMS, and RDF2SOLAP. Table 11 presented the results for RDF2SOLAP. In Table 13 we repeat the results from RDF2SOLAP to present together with two other non-SW tools.

As mentioned earlier, the GIS tool does not use explicit relations between the members but employs spatial joins. Therefore, results for detecting explicit relation algorithms in GIS column are marked as N/A. Both RDBMS and RDF2SOLAP detected same number of *within* relations (39,334), between farm states and parishes (point-polygon). However, RDF2SOLAP detected 47% more *within* relations (2,046) than RDBMS detected (785), between parishes and drainage areas (polygon-polygon).

Table 13: Comparisons of number of topological relations found in each tool (f.s. = farm states, p. = parishes, d.a. = drainage areas) - Adapted from [14]

		TOOLS		
		GIS	**RDBMS**	**RDF2SOLAP**
`detectSpatialHS` **(p. – d.a.)**	*intersects*	N/A	1,897	636
	within	N/A	785	2,046
`detectFactLevelR.` **(f.s.– p.)**	*within*	N/A	39,334	39,334
`discoverSpatialHS` **(p. – d.a.)**	*intersects*	2,556	2,802	1,088
	within	1,039	785	3,392
`discoverFactLevelR.` **(f.s. – p.)**	*within*	39,805	39,984	39,998
`discoverFactLevelR.` **(f.s. – d.a.)**	*within*	39,441	39,845	39,845

Similarly, for discovering spatial hierarchy steps between parishes and drainage areas (polygon-polygon), RDF2SOLAP discovered 47% more *within* relations than GIS tool and 54% more than RDBMS. This is due to the generalization of the polygon data in bounding boxes in RDF2SOLAP, where, in native spatial RDBMS and GIS tool, multi-part polygon data is processed in its original format. In this case, the GIS tool presents the most accurate

results (2,556 *intersects* relations) and RDBMS presents similar results (2,802 *intersects* relations) to the GIS tool with 8% difference.

For discovering fact level relations between farm states and parishes (point-polygon), GIS tool discovered 39,805 *within* relations, RDBMS discovered 39,984 *within* relations, and RDF2SOLAP discovered 39,998 *within* relations, which is only 0,03% more than RDBMS and 0,4% more than the GIS tool. For the same algorithm between farm states and drainage areas, RDBMS and RDF2SOLAP discovered the same number of *within* relations (39,845), and GIS tool discovered 39,441 *within* relations, which is 1% less than RDF2SOLAP and RDBMS, where all the tools present very similar results.

We summarize the topological relations between farm states, parishes and drainage areas on the map given in Figure 27. The results for parish-drainage area relations are grouped in the legend with the green scale from 1 to 5, where 1 with white symbol shows the number of *within* relations (1st tile of the figure), and the green symbols represent the *intersects* relations with 2, 3, 4, and 5 (number of) drainage areas (2nd tile of the figure). The pink symbol represents the 15 parishes that cannot be related to a drainage area with a topological relation (3rd tile of the figure).

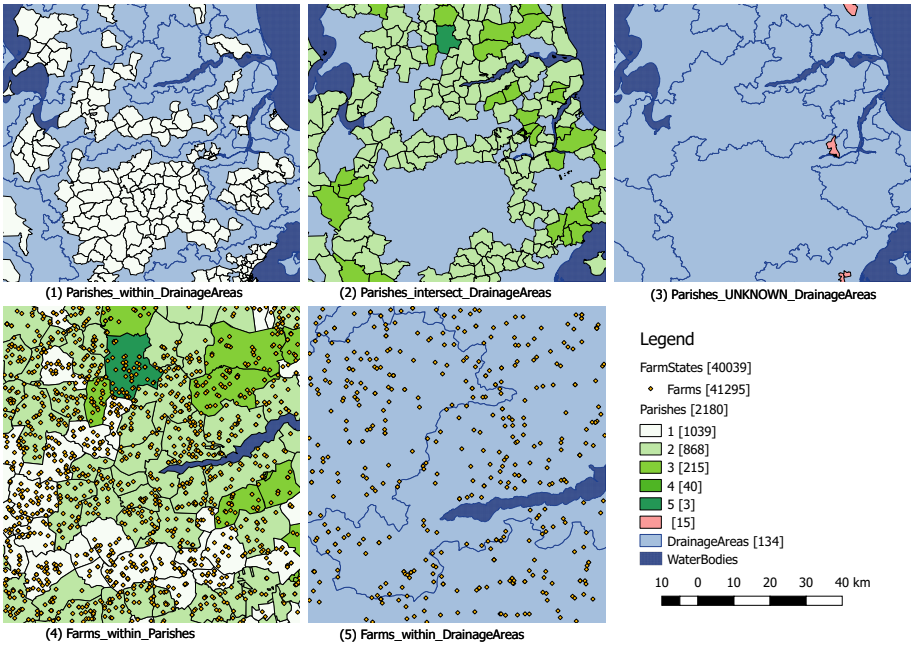


Fig. 27: Topological Relations summarized

Technical Lessons and Summary

In RDF2SOLAP, the implementation results for algorithms (`detectSpatialHS` and `discoverSpatialHS`) that take inputs as polygon geometry type demonstrated a little deviation compared to other two non-SW tools. This can be prevented by using multi-polygon data in its original form instead of generalizing them with bounding-boxes. This can be achieved, either by using the spatial capabilities of a triple store where multi-polygon data and spatial Boolean operators are natively supported or by using a third-party library that can operate with spatial Boolean operators over multi-polygon data natively without generalizing them into bounding boxes.

On the other hand, RDF2SOLAP demonstrated accurate results with less than 1% deviation in the resulting topological relations compared to RDBMS and GIS tool for algorithms that take point-polygon data types as inputs (`detectFactLevelRelations` and `discoverFactLevelRelations`).

In terms of productivity and performance, RDF2SOLAP significantly reduces the overhead work to operate with SW data compared to in-house non-SW proprietary platforms (RDBMS and GIS) by 2 to 3 order of magnitude for spending significantly less time than RDBMS and the GIS tool. Further spatial improvements to the underlying semantic web technologies (triple stores and javascript libraries) can facilitate an improved development environment for RDF2SOLAP with better coverage and accuracy. Even though the spatial support of triple stores is increasing, studies show that there still exist challenges and restrictions in supporting common standards (e.g., GeoSPARQL) and full coverage of complex spatial data types in the current state of the most common well-known triple stores [16, 17]. Improvements to the RDF2SOLAP query processing times can be achieved as well by utilizing spatial indexes on the RDF data in triple stores that support the spatial indexes or by building R-tree in memory on Node.js.

7 Conclusion and Summary of Contributions

The thesis provides the latest state of the art methods for modeling, annotating and querying spatial data warehouses on the Semantic Web. The thesis also provides algorithms for translating high-level SOLAP operators to SPARQL queries and spatial multi-dimensional enrichment algorithms for RDF data cubes on the Semantic Web. In the following, we first give the conclusion of the thesis with the rationale of the papers given in Sections 2-6 and finally summarize the contributions.

In order to investigate the applicability of geo-semantic data warehouses, Paper A presents a use case study (GovAgriBus) on spatial and governmental data domains on the Semantic Web, which is published following common RDF data principles, and queried with aggregate queries in SPARQL, which

are essential query structure for data warehouses. Aggregate query templates are evaluated with different strategies than the native RDF strategy, where native RDF is the most efficient strategy in answering aggregated queries. The use case GovAgriBus was not modeled with multi-dimensional concepts, so that cannot be considered as a basis to geo-semantic data warehouses, yet proofs an important point on the applicability of aggregate queries and basic spatial functions in SPARQL. Therefore, Paper B proposes QB4SOLAP vocabulary (v1) for modeling geo-semantic data warehouses with spatial and multi-dimensional concepts as snowflake schema and publish to the semantic web. In Paper B, QB4SOLAP vocabulary is validated with a realistic use case with spatial concepts. In order to strengthen our case and validation of QB4SOLAP, Paper E presents a geo-semantic data warehouse from real-world governmental, spatial, and environmental data domains, which is modeled and published with QB4SOLAP (v2). Before QB4SOLAP, modeling and annotating the spatial MD concepts such as spatial level attributes, spatial measures with spatial aggregate functions, spatial hierarchies and spatial hierarchy steps with topological relations, etc. were not possible with existing semantic web ontologies and vocabularies. Explicit annotation of these spatial multi-dimensional concepts is essential in a spatial data warehouse in order to query with advanced spatial and analytical (a.k.a. SOLAP) queries.

Paper C improves the QB4SOLAP vocabulary to its latest stable version (v2) and presents a foundation of spatial data warehouses on the semantic web with SOLAP to SPARQL generator algorithms, which are designed using the formal semantics of spatial multi-dimensional RDF concepts from QB4SOLAP. SOLAP to SPARQL generator algorithms are implemented with the GeoSemOLAP tool in Paper D, and tested as individual and as a complex nested set of SOLAP operations. Semantic Web is a new outlet for BI and DW users who would like to query geo-semantic data warehouses with spatial OLAP operations. Native RDF query language (SPARQL) has a complicated syntax for new users. SOLAP to SPARQL generation algorithms and GemSemOLAP lifts the entry barrier for decision-makers and BI users to utilize geo-semantic data warehouses by providing an intuitive user interface, where users can interactively build the SOLAP queries (e.g., s-slice, s-roll-up, etc.) from drop-down menus and interactive maps. This has become possible as a result of QB4SOLAP and its formal semantics, and SOLAP to SPARQL query generation algorithms. The algorithms cover the most common SOLAP operators; s-slice, s-dice, s-roll-up, and s-drill-down with a generic approach where new algorithms can be introduced using the QB4SOLAP semantics and helper functions such as roll-up-path (that finds the path-shaped join of triples with relations from lowest granularity fact members to higher granularity level members).

It is important to note that, to efficiently query geo-semantic data warehouses with SOLAP operators, modeling and annotating the spatial MD

concepts of RDF data with QB4SOLAP is necessary. SOLAP operators use the explicit semantics of QB4SOLAP defined over a use case data, that has a multi-dimensional model designed as snowflake schema. For example, many-to-many (*n-n*) relations between spatial levels along a dimension implicates each spatial level member should have direct relations defining the nature of the relation, such as annotated with topological relations (e.g., intersects, within, etc.). This way, we can ensure the roll-up paths are created correctly in SOLAP operations.

Semantic Web accommodates many, multi-dimensional, statistical, and spatial data sets that may not have been annotated with QB4SOLAP, thus the SOLAP operators cannot be utilized over these data sets. Paper F proposes an RDF2SOLAP enrichment framework to address this issue by annotating existing RDF data with QB4SOLAP, which is feasible for multi-dimensional modeling and has spatial values that could benefit from SOLAP operations. RDF2SOLAP enrichment framework is built with two phases: hierarchical enrichment and factual enrichment. In each phase, there are two algorithms that find explicit and implicit relations to annotate spatial multi-dimensional concepts (i.e., hierarchy steps or fact-level relations), which are composing a roll-up-path. Explicit relations are interpreted from the existing RDF data by referential integrity keys and join of triples between level member URIs by explicit predicates. By utilizing the explicit links between members, RDF2SOLAP enrichment algorithms detect the precise topological relation between spatial members and annotate in the output (e.g., within, intersects). Implicit relations are not defined, thus the enrichment algorithms discover all possible topological relations between spatial members that can be annotated within a dimension hierarchy and outputs the annotated results. Factual enrichment algorithms address both fact-base level spatial annotations and fact-parent level annotations if there is an *n-n* cardinality relation between a child level and parent level. This way, a drill-down from the parent level member to the correct lowest member (fact member) becomes possible through the explicitly annotated topological relation. Finally, factual enrichment algorithms re-defines the fact schema from the spatial concepts derived during enrichment, such as spatial measures and their aggregate functions.

Quantitative evaluation of the enrichment algorithms demonstrates a significant improvement for the BI users on the SW, since RDF2SOLAP handles the data extract, query processing, and annotation at a reasonable time frame at once, on the other hand, users would need to spend significant effort to get similar results from operational RDBMS and GIS environments (as they have to manually extract the RDF data set from endpoints, convert the data to native formats of the systems, prepare and process the queries to derive topological relations, and annotate enrichment results of the data). Qualitative evaluation of the enrichment algorithms presents comparable results in

terms of the number of detected/discovered topological relations against the RDBMS and GIS systems, where there is a room for improvement in multi-polygon data structures. Overall, RDF2SOLAP is built as an important key to the Semantic Web for annotating geo-semantic data warehouses on the fly directly over RDF data.

To sum up, the papers in the thesis make the following contributions.

- Paper A [3] presents the best practices for publishing Danish Agricultural Open Data sets on the Semantic Web as an initial effort to publish open governmental data sets in RDF format in Denmark. The use case is selected from non-trivial open governmental data sets covering agricultural, business, and geographical domains. The paper also delivers different extract, transform, and load (ETL) strategies for preparing the Semantic Web data in RDF format from the heterogeneous sources in several file formats. Finally, the paper introduces different query templates for aggregate queries and standard queries to evaluate different query processing scenarios with native RDF, relational, and virtual strategies. The evaluation of the query processing suggests that the query performance for aggregated query templates on native RDF is significantly faster than the other strategies, which is a promising step for semantic web data warehousing since aggregate queries are fundamental in multi-dimensional models and in data warehousing. Therefore, this paper provides a motivation for the other papers.

- Paper B [12] proposes a multi-dimensional vocabulary - QB4SOLAP for modeling and annotating spatial data warehouses on the Semantic Web. In order to propose a state of the art vocabulary for spatial data warehouses (on the SW), the paper thoroughly presents the related work and recent relevant vocabularies and semantic web technologies. As a result, the paper introduces the QB4SOLAP vocabulary, as an extension of the most recent non-spatial multi-dimensional vocabulary for modeling traditional data warehouses on the SW - QB4OLAP. The proposed QB4SOLAP vocabulary is applied to a non-trivial realistic use case, which has spatial data. Together with the use case, the notion of Spatial OLAP (SOLAP) operators are introduced on the Semantic Web and how to write SOLAP queries in SPARQL are given in detail with examples.

- Paper C [15] extends Paper B [12] by giving full formal definitions of the multi-dimensional spatial concepts from QB4SOLAP in RDF format. The key concepts of QB4SOLAP vocabulary such as spatial dimensions, spatial levels, spatial hierarchies and hierarchy steps, spatial measures, spatial aggregate functions, topological relations, etc. and moreover common SOLAP operators are defined formally in the paper,

which can easily be referenced and encapsulated in algorithms. The paper provides, algorithms for generating individual and nested SOLAP queries in SPARQL from high-level multi-dimensional expressions. The (SPARQL query) generation algorithms allow data warehouse users to query geo-semantic data warehouses without knowledge of RDF/SPARQL.

- Paper D [13] introduces the GeoSemOLAP tool, which is implemented using the generation algorithms from Paper C [15]. GeoSemOLAP allows users to query SW with individual or nested SOLAP operations, without writing a single line of SPARQL query. GeoSemOLAP provides end-users interactive maps, where DW users can click on a map to select spatial attributes or levels in a SOLAP operator in addition to the possibility of creating the high-level SOLAP query (e.g., spatial *slice, dice, roll-up*) from drop-down menus showing the multi-dimensional schema elements.

- Paper E [11] applies QB4SOLAP on a non-trivial open governmental and spatial data collection from Danish governmental organizations including environmental, agricultural, geographical, and business data sets. The paper also presents the challenges and best practices for publishing geo-semantic data warehouses with discussions and perspectives. The published use-case as a geo-semantic data warehouse with QB4SOLAP vocabulary is also validated in the paper with non-trivial nested SOLAP queries in SPARQL.

- Paper F [14] addresses the issue on lacking spatial multi-dimensional annotations on the Semantic Web and proposes an RDF2SOLAP enrichment framework to enrich existing RDF data cubes. RDF2SOLAP enrichment model introduces the enrichment process with hierarchical enrichment algorithms and factual level enrichment algorithms. Hierarchical enrichment algorithms cover detecting explicit relations between defined level members with non-spatial hierarchy steps and discovering implicit relations without direct links between the level members. Both implicit and explicit relations are annotated as topological relations and enriched in the data set for providing essential analytical insight. Factual enrichment algorithms cover the previous two similar detecting explicit relations and discovering implicit relations approach between fact and level members. In addition to this, factual enrichment is capable of redefining the fact schema in an automated from the outcome of the first two algorithms. The framework is applied to a non-trivial use case and the experimental evaluation findings suggest that the proposed framework demonstrates efficient handling and processing of the enrichment and annotation algorithms directly on the RDF data, and it

also presents competitive quality to get similar results from non-SW spatial query tools and products.

In conclusion, QB4SOLAP is proven as the spatial multi-dimensional vocabulary for modeling geo-semantic data warehouses, so that the advanced analytical queries with spatial perspectives over Semantic Web data becomes possible with SOLAP operators, which were not possible before. By using the proposed SOLAP to SPARQL generation algorithms and formal QB4SOLAP semantics, the GeoSemOLAP tool allows inexperienced semantic web users to query geo-semantic data warehouses with SOLAP operations without writing a single line of SPARQL query. To derive spatial, multi-dimensional and analytical perspectives from the existing RDF datasets, the RDF2SOLAP enrichment framework with factual and hierarchical enrichment phase algorithms provides a considerably easy way of annotating geo-semantic data warehouses directly from existing RDF data sets.

8 Future Work

The rationale and result of the thesis is to research and present the best practices, tools, and algorithms for modeling, annotating, and, querying geo-semantic data warehouses. The thesis also opens several interesting research areas.

The research presented in Section 2 for publishing Danish governmental, agricultural data sets as Linked Open Data can be automated by following the best practices every year. Danish Ministry of Agriculture updates with the most recent agricultural data yearly. Including a temporal basis of updating the GovAgriBus Denmark data set would require as well linking and annotating the use case data with the temporal vocabularies such as Time Ontology of W3C. In this line of work, new concepts and standards can be developed for easily and efficiently creating spatial and spatio-temporal relationships to link and publish agricultural and spatial data that is changing on a temporal basis.

In order to enable business intelligence analytics on the Semantic Web for spatial data, QB4SOLAP is introduced in Section 3 with a set of common SOLAP operators and we described how to query the SW by using these SOLAP operators, which can be translated into SW query language - SPARQL by GeoSemOLAP tool (Section 4). Extending the scope of these analytical operators with more complex ones such as spatial drill-across and spatial aggregation over spatial measures would be a challenging and interesting research direction by pushing the technical limitations of the Semantic Web on representing and analytical querying of spatial data. Finally, it would be also interesting to optimize and increase the efficiency of our SOLAP query tech-

niques such as materialization and optimizing the layout of query templates, or by employing federated processing of spatial analytical queries.

By using QB4SOLAP, Section 5 demonstrates a proof of concept non-trivial governmental data use case that is published on the Semantic Web. A semantic ETL tool that can semi-automatically generate LOD data, which is annotated with QB4SOLAP would be a handy tool to publish geo-semantic cubes directly from governmental public open data portals. In Section 6, we introduced RDF2SOLAP multi-dimensional enrichment process, which takes non-spatial QB and QB4OLAP cubes as a basis on the SW. Developing methods and algorithms for a fully functional geo-semantic ETL process from heterogeneous non-SW geo data portals can complement the RDF2SOLAP enrichment process by extending the source basis to non-SW spatial data. Moreover, semantic extensions can also be introduced on top of RDF2SOLAP enrichment process by linking the base data with external geographical vocabularies and data sets. External geographical data on the SW can bring new dimensional perspectives over the existing data such as new spatial hierarchy levels and new spatial level members.

Another line of interesting future work around RDF2SOLAP enrichment algorithms can be creating a comprehensive benchmark test by employing the algorithms to query SPARQL endpoints from different RDF stores employing the spatial capabilities (e.g., spatial indexes) of the RDF stores. Moreover, it is important to develop query optimization techniques for OLAP queries on semantic DW RDF data, similar to the ones developed for cubes and XML data [24, 25, 37]. Furthermore, to achieve scalable querying and runtime optimization, new research directions can be taken with binary serialization of the QB4SOLAP RDF data such as header dictionary triples (HDT), which is a compact data structure that can be compressed and kept in-memory, thus it enables high performance (and also concurrent) querying.

The contributions of the thesis can be employed to inspire spatio-temporal data warehouses on the semantic web, where the temporal dimension of the data warehouses and its challenges can be addressed in a similar way to the spatial dimension and handling its challenges. The analogy to the spatial Boolean algebra (with topological relations) in spatial data warehouses is *calculus for temporal reasoning* (with Allen's interval algebra [2]) in temporal data warehouses. This would bring new modeling and annotation opportunities on top of QB4SOLAP using the existing temporal ontologies. Furthermore, it would be interesting to extend the coverage of our algorithms (query generator and enrichment) to handle highly dynamic spatio-temporal queries.

References

[1] "The Friend of a Friend (FOAF) Project," http://www.foaf-project.org/.

[2] J. Allen, "Maintaining Knowledge about Temporal Intervals," 1983.

[3] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Semantic Technology: 4th Joint International Semantic Technology Conference (JIST'14)*, vol. 8943. Springer, 2014, pp. 178–186, https://dx.doi.org/10.1007/978-3-319-15615-6_13.

[4] A. B. Andersen, N. Gur, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *DBTR-35*. Aalborg University, 2014, p. 16, http://dbtr.cs.aau.dk/DBPublications/DBTR-35.pdf.

[5] J. B. Arendt, "Denmark releases its digital raw material," http://uk.fm.dk/news/, Ministry of Finance of Denmark, Denmark Ministry of Finance, October 2012.

[6] Y. Bédard, E. Bernier, S. Larrivée, M. Nadeau, M. Proulx, and S. Rivest, "Spatial OLAP," in *Forum annuel sur la RD, Géomatique VI: Un monde accessible*, 1997, pp. 13–14.

[7] W.-D. Brickley, "W3C Semantic Web Interest Group: Geo," http://www.w3.org/2003/01/geo/wgs84_pos, www.wgs84.com.

[8] E. Edoh-Alove, S. Bimonte, and F. Pinet, "An UML Profile and SOLAP Datacubes Multidimensional Schemas Transformation Process for Datacubes Risk-Aware Design," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 11, no. 4, pp. 64–83, 2015, https://dx.doi.org/10.4018/ijdwm.2015100104.

[9] M. J. Egenhofer and J. Herring, "A mathematical framework for the definition of topological relationships," in *Fourth international symposium on spatial data handling*. Zurich, Switzerland, 1990, pp. 803–813.

[10] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web using QB4OLAP," in *Data Warehousing and Knowledge Discovery (DaWaK'14)*, vol. 8646. Springer, 2014, pp. 45–56, https://dx.doi.org/10.1007/978-3-319-10160-6_5.

[11] N. Gür, K. Hose, T. B. Pedersen, and E. Zimányi, "Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP," in *Semantic Technology: 6th Joint International Semantic Technology Conference (JIST'16)*, vol. 10055. Springer, 2016, pp. 287–304, https://dx.doi.org/10.1007/978-3-319-50112-3_22.

[12] N. Gür, K. Hose, E. Zimányi, and T. B. Pedersen, "Modeling and Querying Spatial Data Warehouses on the Semantic Web," in *Semantic Technology: 5th Joint International Semantic Technology Conference (JIST'15)*, vol. 9544. Springer, 2015, pp. 1–20, https://dx.doi.org/10.1007/978-3-319-31676-5_1.

[13] N. Gür, J. Nielsen, K. Hose, and T. B. Pedersen, "GeoSemOLAP: SOLAP on the Semantic Web Made Easy," in *Proceedings of the 26th International Conference Companion on World Wide Web (WWW'17)*. ACM, 2017, https://dx.doi.org/10.1145/3041021.3054731.

[14] N. Gür, T. B. Pedersen, and K. Hose, "Multidimensional Enrichment of Spatial RDF Data for SOLAP," *Semantic Web Journal*, vol. under submission, 2019, http://www.semantic-web-journal.net/content/multidimensional-enrichment-spatial-rdf-data-solap-0.

[15] N. Gür, T. B. Pedersen, E. Zimányi, and K. Hose, "A Foundation for Spatial Data Warehouses on the Semantic Web," *Semantic Web Journal*, vol. 9, no. 5, pp. 557–587, 2018.

[16] W. Huang, S. A. Raza, O. Mirzov, and L. Harrie, "Assessment and benchmarking of spatially enabled rdf stores for the next generation of spatial data infrastructure," *ISPRS International Journal of Geo-Information*, vol. 8, no. 7, p. 310, 2019.

[17] T. Ioannidis, G. Garbis, K. Kyzirakos, K. Bereta, and M. Koubarakis, "Evaluating geospatial rdf stores using the benchmark geographica 2," *arXiv preprint arXiv:1906.01933*, 2019.

[18] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *The Semantic Web: ESWC 2012 Satellite Events*, vol. 7540. Springer, 2012, pp. 87–101, https://dx.doi.org/10.1007/978-3-662-46641-4_7.

[19] T. B. Lee, "Design issues," W3C, July 2006, http://www.w3.org/DesignIssues/LinkedData.html.

[20] I. V. Lopez, R. T. Snodgrass, and B. Moon, "Spatiotemporal aggregate computation: A survey," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 17, no. 2, pp. 271–286, 2005, https://doi.org/10.1109/TKDE.2005.34.

[21] Danish Ministry of the Environment, "Consolidated Act on Livestock Farming Environmental Approvals," 2012, http://eng.mst.dk/media.

[22] Nitrates Directive, "Danish nitrate action programme 2008-2015 regarding the nitrates directive; 91/676/eec," http://eng.mst.dk/media/mst/Attachments/DanishNitrateActionProgramme2008201507092012.pdf, Nitrates Directive, Tech. Rep., 2012.

[23] Microsoft, "Multidimensional Data Expressions," https://docs.microsoft.com/en-us/sql/analysis-services/multidimensional-models/mdx.

[24] D. Pedersen, J. Pedersen, and T. B. Pedersen, "Integrating xml data in the targit olap system," in *Proceedings. 20th International Conference on Data Engineering*. IEEE, 2004, pp. 778–781.

[25] D. Pedersen, K. Riis, and T. B. Pedersen, "Query optimization for OLAP-XML federations," in *Proceedings of the 5th International Workshop on Data Warehousing and OLAP (DOLAP'02)*. ACM, 2002, pp. 57–64.

[26] T. B. Pedersen and N. Tryfona, "Pre-aggregation in Spatial Data Warehouses," in *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD'01)*. Springer, 2001, pp. 460–478, http://dx.doi.org/10.1007/3-540-47724-1_24.

[27] D. A. Randell, Z. Cui, and A. G. Cohn, "A spatial logic based on regions and connection," in *Principles of Knowledge Representation and Reasoning*, vol. 92, 1992, pp. 165–176.

[28] G. Rojas, G. Giannopoulos, and J. J. L. Daniel Hladky, "Managing Geospatial Linked Data in the GeoKnow Project," in *The Semantic Web in Earth and Space Science. Current Status and Future Directions*, vol. 20. IOS Press, 2015, p. 51.

[29] O. Software, "Virtuoso RDF Views – Getting Started Guide," June 2007, http://www.openlinksw.co.uk/virtuoso/Whitepapers/pdf/Virtuoso_SQL_to_RDF_Mapping.pdf.

[30] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "LinkedGeoData: A Core for a Web of Spatial Open Data," *Semantic Web Journal (SWJ)*, vol. 3, pp. 333–354, 2012, https://dx.doi.org/10.3233/SW-2011-0052.

[31] A. I. M. Standards, "AGROVOC Linked Open Data," http://aims.fao.org/aos/agrovoc/.

[32] A. Vaisman and E. Zimányi, "Spatial data warehouses," in *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[33] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, and C. Thomsen, "QB2OLAP: Enabling OLAP on Statistical Linked Open Data," in *32nd IEEE International Conference on Data Engineering*, 2016, pp. 1346–1349.

[34] W3C, "Data Cube Implementations," 2014, https://www.w3.org/2011/gld/wiki/Data_Cube_Implementations.

[35] X. Wang, S. Staab, and T. Tiropanis, "ASPG: generating OLAP queries for SPARQL benchmarking," in *Semantic Technology - 6th Joint International Conference, JIST 2016, Singapore, Singapore, November 2-4, 2016, Revised Selected Papers*, 2016, pp. 171–185. [Online]. Available: https://doi.org/10.1007/978-3-319-50112-3_13

[36] M. Wick, "GeoNames Ontology," http://www.geonames.org/ontology/documentation.html.

[37] X. Yin and T. B. Pedersen, "Evaluating xml-extended olap queries based on physical algebra," *Journal of Database Management (JDM)*, vol. 17, no. 2, pp. 85–116, 2006.

Part II

Papers

Paper A

Publishing Danish Agricultural Government Data as Semantic Web Data

Alex B. Andersen, Nurefşan Gür, Katja Hose, Kim A. Jakobsen, and Torben Bach Pedersen

The paper has been published in the
Proceedings of the 4th Joint International Semantic Technology Conference
Vol. 8943, pp. 178–186, 2014. DOI: 10.1007/978-3-319-15615-6_13

Abstract

Recent advances in Semantic Web technologies have led to a growing popularity of the Linked Open Data movement. Only recently, the Danish government has joined the movement and published several datasets as Open Data. These raw datasets are difficult to process automatically and combine with other data sources on the Web. Hence, our goal is to convert such data into RDF and make it available to a broader range of users and applications as Linked Open Data. In this paper, we discuss our experiences based on the particularly interesting use case of agricultural data as agriculture is one of the most important industries in Denmark. We describe the process of converting the data and discuss the particular problems that we encountered with respect to the considered datasets. We additionally evaluate our result based on several queries that could not be answered based on existing sources before.

1 Introduction

In recent years, more and more structured data has become available on the Web, driven by the increasing popularity of both the Semantic Web and Open Data movement, which aim at making data publicly available and free of charge. Several governments have been driving forces of the Open Data movement, most prominently `data.gov.uk` (UK) and `data.gov` (USA), which publish Open Data from departments and agencies in the areas of agriculture, health, education, employment, transport, etc.

The goal is to enable collaboration, advanced technologies, and applications that would otherwise be impossible or very expensive, thus inspiring new services and companies. Especially for governments, it is important to inspire novel applications, which will eventually increase the wealth and prosperity of the country.

While publication of raw data is a substantial progress, the difficulty in interpreting the data as well as the heterogeneity of publication formats, such as spreadsheets, relational database dumps, and XML files, represent major obstacles that need to be overcome [1–3] – especially because the schema is rarely well documented and explained for non-experts. Furthermore, it is not possible to evaluate queries over one or multiple of these datasets.

The Linked (Open) Data movement (`http://linkeddata.org/`) encourages the publication of data following the Web standards along with *links* to other data sources providing semantic context to enable easy access and interpretation of structured data on the Web.

Hence, publishing data as Linked Data (LD) [4, 5] entails the usage of certain standards such as HTTP, RDF, and SPARQL as well as HTTP URIs as entity identifiers that can be dereferenced, making LD easily accessible on the Web. RDF allows formulating statements about resources, each statement consists of subject, predicate, and object – referred to as a triple. Extending the dataset and adding new data is very convenient due to the self-describing nature of RDF and its flexibility.

In late 2012, the Danish government joined the Open Data movement by making several raw digital datasets [6] freely available. Among others, these datasets cover transport, tourism, fishery, companies, forestry, and agriculture. To the best of our knowledge, they are currently only available in their raw formats and have not yet been converted to LD.

We choose agriculture as a use case, as it is one of the main sectors in Denmark, with 66% of Denmark's land surface being farmland[1]. Thus, there is significant potential in providing free access to such data and enabling efficient answering of *sophisticated* queries over it.

In this paper, we show how we made Danish governmental Open Data

[1] `http://www.statistikbanken.dk/AREALAN1`

available as LD and evaluate the challenges in doing so. Our approach is to transform the agricultural datasets into RDF and add explicit relationships among them using links. Furthermore, we integrate the agricultural data with company information, thus enabling queries on new relationships not contained in the original data.

This paper presents the process to transform and link the data as well as the challenges encountered and how they were met. It further discusses how these experiences can provide guidelines for similar projects. We developed our own ontology while still making use of existing ontologies whenever possible. A particular challenge is deriving spatial containment relationships not encoded in the original datasets. For a detailed discussion about the whole process, we refer the reader to the extended version of this paper [7]. The resulting LOD datasets are accessible via a SPARQL endpoint (`http://extbi.lab.aau.dk/sparql`) as well as for download (`http://extbi.cs.aau.dk/`).

The remainder of this paper is structured as follows; Section 2 describes our use case datasets and discusses the main challenges. Then, Section 3 describes the process and its application to the use case. Section 4 evaluates alternative design choices, while Section 5 concludes and summarizes the paper.

2 Use Case

We have found the agricultural domain to be particularly interesting as it represents a non-trivial use case that covers spatial attributes and can be extended with temporal information. By combining the agricultural data with company data, we can process and answer queries that were not possible before as the original data was neither linked nor in a queryable format.

Late 2012, the Ministry of Food, Agriculture, and Fisheries of Denmark (FVM) (`http://en.fvm.dk/`) made geospatial data of all fields in Denmark freely available – henceforth we refer to this collection of data as *agricultural data*. This dataset combined with the *Central Company Registry (CVR) data* (`http://cvr.dk/`) about all Danish companies allows for evaluating queries about fields and the companies owning them. In total, we have converted 5 datasets provided by FVM and CVR into Linked Open Data. We downloaded the data on October 1, 2013 from FVM [8] and from CVR.

Agricultural Data. The agricultural data collection is available in Shape format [9], this means that each *Field*, *Field Block*, and *Organic Field* is described by several coordinate points forming a polygon.

Field. The Field dataset has 9 attributes and contains all registered fields in Denmark. In total, this dataset contains information about 641,081 fields.

Organic Field. This dataset has 12 attributes and contains information

about 52,060 organic fields. The dataset has attributes that we can relate to the company data, i.e., the CVR attribute is unique for the owner of the field and references the CVR dataset that we explain below. The fieldBlockId attribute describes to which “Field Block" a field belongs to.

Field Block. The Field Block dataset has 12 attributes for 314,648 field blocks and contains a number of fields [10]. Field Blocks are used to calculate the funds the farmers receive in EU area support scheme.

Central Company Registry (CVR) Data. The CVR is the central registry of all Danish companies and provides its data in CSV format. There are two datasets available that we refer to as *Company* and *Participant*.

Company. This dataset has 59 attributes [11] and contains information, such as a company’s name, contact details, business format, and activity for about more than 600,000 companies and 650,000 production units.

Participant. This dataset describes the relations that exist between a participant and a legal unit. A participant is a person or legal unit that is responsible for a legal unit in the company dataset, i.e., a participant is an owner of a company. The Participant dataset describes more than 350,000 participants with 7 attributes.

The use case data comes in different formats and contains only a few foreign keys. Further, there is little cross-reference and links between the datasets and no links to Web sources in general. Spatial relationships are even more difficult to represent in the data and querying data based on the available polygons is a complex problem. In particular, to enable queries that have not been possible before, we cleanse and link the (Organic) Field datasets to the Company dataset so that we can query fields and crops of companies related to agriculture. The particular challenges that we address are:

- Disparate data sources without common format
- Lack of unique identifiers to link different but related data sources
- Language (Danish)
- Lack of ontologies and their use

3 Data Annotation and Reconciliation

In this section, we outline the process that we followed to publish the datasets described in Section 2. The complete procedure with its main activities is depicted in Fig. A.1.

All data in the data repository undergoes an iterative integration process consisting of several main activities:

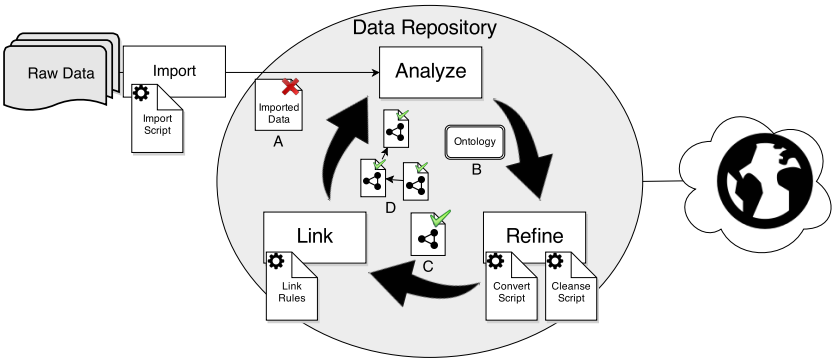


Fig. A.1: Process overview

Import: Extract the data from the original sources

Analyze: Gain an understanding of the data and create an ontology

Refine: Refine the source data by cleansing it and converting it to RDF

Link: Link the data to internal and external data

Data that has been through the integration process at least once may be published and thus become Linked Open Data that others can use and link to. In the remainder of this section, we will discuss these steps in more detail.

Import. The raw data is extracted from its original source into the repository and stored in a common format such that it is available for the later activities. The concrete method used for importing a dataset depends on the format of the raw data. The agriculture datasets and CVR datasets introduced in Section 2 are available in Shape and CSV formats. Shape files are processed in ArcGIS[2] to compute the spatial joins of the fields and organic fields, thus creating foreign keys between the datasets. As the common format we use a relational database.

Analyze. The goal of this step is to acquire a deeper understanding of the data and formalize it as an ontology. As a result of our analysis we constructed a URI scheme for our use case data based on Linked Data Principles [4]. We strive to use existing ontologies as a base of our own ontologies. To do this, we make use of predicates such as `rdfs:subClassOf`, `rdfs:subPropertyOf`, and `owl:equivalentClass`, which can link our classes and properties to known ontologies. Fig. A.2 provides an overview of the ontology that we developed for our use case with all classes and properties. All

[2] `http://www.esri.com/software/arcgis`

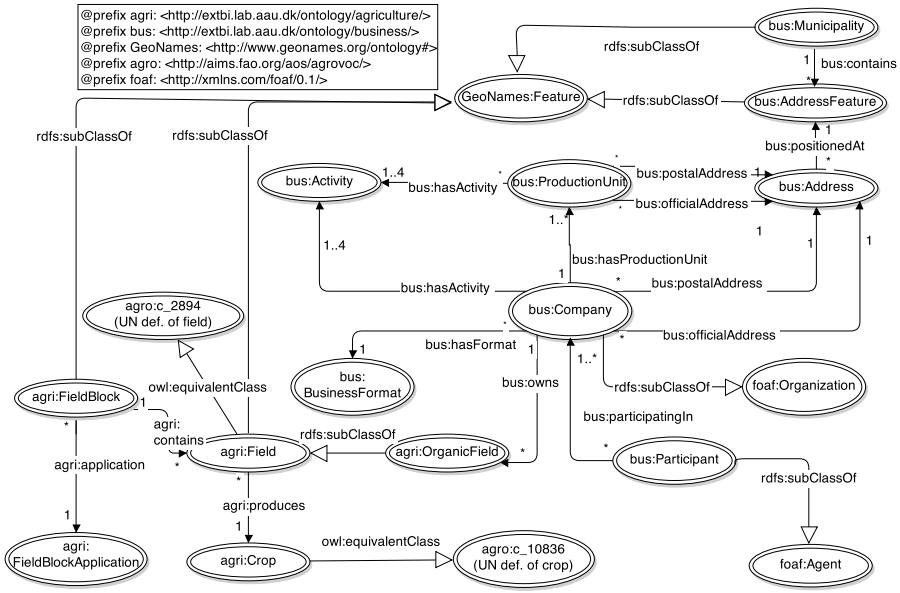


Fig. A.2: Overview of the ontology for our use case

arrows are annotated with predicates. The arrows with black tips represent relations between the data instances. The arrows with white tips represent relations between the classes.

In short, we designed the ontology such that a Field is contained within a Field Block, which is expressed with the property `agri:contains` and is determined by a spatial join of the data. Organic Field is a subclass of Field and therefore transitively connects Field to Company. Field is also defined as being equivalent to the UN's definition of European fields from the AGROVOC [12] vocabulary. In addition we make use of other external ontologies and vocabularies, such as *GeoNames* [13], *WGS84* [14], and *FOAF (Friend of a Friend)* [15].

Refine. The Refine activity is based on the understanding gained in the Analyze activity and consists of data cleansing and conversion. Fig. A.1 illustrates the data cleansing process where imported data and ontologies are used to produce cleansed data.

In our use case, we implemented data cleansing by using views that filter out inconsistent data as well as correct invalid attribute values, inconsistent strings, and invalid coordinates. Then we use Virtuoso Opensource [16] mappings to generate RDF data.

Link. The Link activity consists of two steps: internal linking and external linking, which converts the refined data into integrated data. The Link ac-

tivity materializes the relationships between concepts and classes identified in the Analyze activity as triples. The example below shows our internal linking of the Field and the Field Block classes using the `geonames:contains` predicate.

```
@prefix agri:<http://extbi.lab.aau.dk/ontology/agriculture/>.
@prefix geonames:<http://www.geonames.org/ontology#>.

agri:contains rdf:type owl:ObjectProperty ;
rdfs:domain agri:FieldBlock ;
rdfs:range agri:Field ;
rdfs:subPropertyOf geonames:contains .
```

External linking involves linking to remote sources on instance and ontology level. On the ontology level, this means inserting triples using predicates such as `rdfs:subClassOf`, `rdfs:subPropertyOf`, and `owl:equivalentClass` that link URIs from our local ontology to URIs from remote sources. On instance level, we link places mentioned in the CVR data to equivalent places in GeoNames [13] using triples with the `owl:sameAs` predicate as illustrated in Fig. A.3.

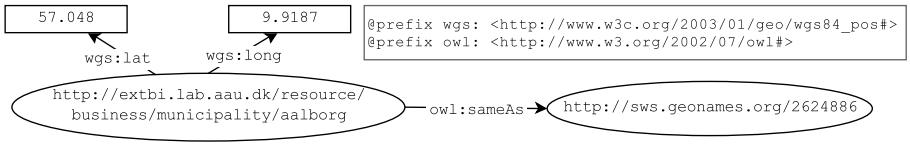


Fig. A.3: External linking on instance level

The overall process has provided us with analyzed, refined, and linked data; in total *32,457,657* triples were created. The result of completing this process is published and registered on `datahub.io`[3]. In case we wish to integrate additional sources, we simply have to reiterate through the process.

4 Experiments

In the following, we first describe three alternative design choices in the materialization of the data. They represent trade-offs between data load time and query time. We then discuss the results of our experimental evaluation, for which we ran an OpenLink Virtuoso 07.00.3203 server on a 3.4 GHz Intel Core i7-2600 processor with 8 GB RAM operated by Ubuntu 13.10, Saucy. The materialization strategies that we have considered are: *Virtual*, *relational*

[3] `http://datahub.io/dataset/govagribus-denmark`

materialization, and *native*. Fig. A.4 shows the different paths that data is traveling on; starting as raw data and ending at the user who issued a query. The solid lines represent data flow during the integration process whereas dashed lines represent data flow at query time.

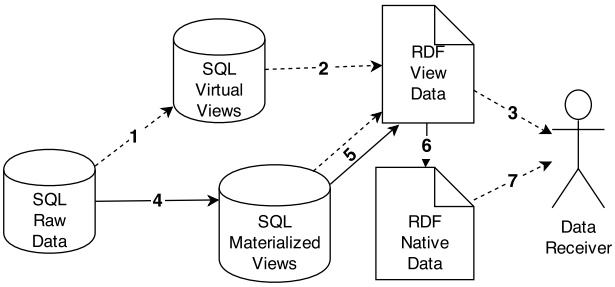


Fig. A.4: Data flow for the materialization strategies

Virtual. In the virtual strategy we perform data cleansing based on SQL views in the relational database. RDF mappings are formulated on top of these cleansing views to make the data accessible as RDF. To increase performance, we create a number of indexes on primary keys, foreign keys, and spatial attributes. In Fig. A.4, using this strategy data flows through the arrows marked with 1, 2, and 3 at query time.

Relational Materialization. Here, we materialize the above mentioned SQL views as relational tables. We create similar indexes as above but on the obtained tables. In Fig. A.4, data flows through arrows 4, 5, and 3 – with 4 during load time and 3 and 5 during query time.

Native RDF. In this strategy, we extract all RDF triples from the materialized views and mappings and load them into a triple store. In Fig. A.4, data flows through arrows 4, 5, and 6 during load time and arrow 7 during query time.

To test our setup, we created a number of query templates that we can instantiate with different entities and that are based on insights in agricultural contracting gained from field experts. Some of them contained aggregation and grouping (Aggregate Query Templates, AQT) others only standard SPARQL 1.0 constructs (Standard Query Templates, SQT). For the virtual and relational materialization strategies we measured the load times for each step during loading – the results are shown in Table A.1. Table A.2 shows the execution times for our query templates on the three materialization strategies. Queries that run into a timeout are marked by a dash. As we can see, the native RDF strategy is faster than the two others, and relational materialized is generally faster than virtual. There is obviously a notable overhead when using views and mappings. On the other hand, the virtual strategy has very

Step	Virtual	Materialized	Native
Data Cleansing	74.92	603.35	603.35
Load Ontology	1.01	1.01	1.01
Load Mappings	8.76	12.35	12.35
Dump RDF	0.00	0.00	4684.82
Load RDF	0.00	0.00	840.04
Total	84.68	616.70	6141.56

Table A.1: Load times in seconds

Query	Virtual	Materialized	Native
AQT 1	5.92	3.39	1.04
AQT 2	13.32	7.00	0.23
AQT 3	10.81	7.70	0.05
AQT 4	–	–	0.14
AQT 5	–	20.37	0.86
SQT 1	–	–	2.35
SQT 2	0.09	0.12	0.10
SQT 3	2188.85	1.81	0.40
SQT 4	6.57	2.35	1.63
SQT 5	–	23.79	3.29
Average	370.93	8.31	1.01

Table A.2: Runtimes in seconds

fast load time compared to the other strategies since no data has to be moved or extracted – in fact, the cleansing is delayed until query time. The relational materialized strategy is one order of magnitude faster in load time than the native strategy as it has less overhead during loading.

We can therefore conclude that the virtual strategy is well suited for rapidly changing data as it has minimum load time, the materialized strategy represents a trade-off between load time and query time and is suitable for data with low update rates, and the native strategy decouples RDF data from the relational data and is very suitable for static data.

5 Conclusion

Motivated by the increasing popularity of both the Semantic Web and the Open (Government) Data movement as well as the recent availability of interesting open government data in Denmark, this paper investigated how to make Danish agricultural data available as Linked Open Data. We chose the most interesting agricultural datasets among a range of options, transformed

them into RDF format, and created explicit links between those datasets by matching them on a spatial level. Furthermore, the agricultural data was integrated with data from the central company registry. All these additional links enable queries that were not possible directly on the original data. The paper presents best practices and a process for transforming and linking the data. It also discusses the challenges encountered and how they were met. As a result, we not only obtained an RDF dataset but also a new ontology that also makes use of existing ontologies. A particularly interesting challenge was how to derive spatial containment relationships not contained in the original datasets because existing standards and tools do not provide sufficient support. The resulting LOD datasets were made available for download and as a SPARQL endpoint.

References

[1] F. Maali, R. Cyganiak, and V. Peristeras, "A publishing pipeline for linked government data," in *The Semantic Web: ESWC'12*, 2012, pp. 778–792. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30284-8_59

[2] B. Villazón-Terrazas, L. Vilches-Blázquez, O. Corcho, and A. Gómez-Pérez, "Methodological Guidelines for Publishing Government Linked Data," in *Linking Government Data*. Springer New York, 2011, pp. 27–49. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1767-5_2

[3] M. G. Skjæveland, E. H. Lian, and I. Horrocks, "Publishing the Norwegian Petroleum Directorate's FactPages as Semantic Web Data," in *International Semantic Web Conference: ISWC'13*, 2013, pp. 162–177. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-41338-4_11

[4] T. B. Lee, "Design issues," W3C, July 2006, http://www.w3.org/DesignIssues/LinkedData.html.

[5] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, ser. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[6] J. B. Arendt, "Denmark releases its digital raw material," http://uk.fm.dk/news/, Ministry of Finance of Denmark, Denmark Ministry of Finance, October 2012.

[7] A. B. Andersen, N. Gur, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *DBTR-35*. Aalborg University, 2014, p. 16, http://dbtr.cs.aau.dk/DBPublications/DBTR-35.pdf.

[8] A. Ministry of Food and F. of Denmark, "FVM Geodata Download," https://kortdata.fvm.dk/download/index.html.

[9] ESRI, "Shapefile technical description," *An ESRI White Paper*, 1998, http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

[10] D. M. of the Environment, "Markblokkort (datasæt)," http://www.geodata-info.dk/Portal/ShowMetadata.aspx?id=1eb89ebb-f674-4ad1-9e53-d1e252226596.

[11] Erhvervsstyrelsen, "Record layout: Juridiske enheder og P-enheder," http://www.cvr.dk/Site/Resources/Files/Media/RecordlayoutABO110.pdf.

[12] A. I. M. Standards, "AGROVOC Linked Open Data," http://aims.fao.org/aos/agrovoc/.

[13] M. Wick, "GeoNames Ontology," http://www.geonames.org/ontology/documentation.html.

[14] W.-D. Brickley, "W3C Semantic Web Interest Group: Geo," http://www.w3.org/2003/01/geo/wgs84_pos, www.wgs84.com.

[15] "The Friend of a Friend (FOAF) Project," http://www.foaf-project.org/.

[16] O. Software, "Virtuoso RDF Views – Getting Started Guide," June 2007, http://www.openlinksw.co.uk/virtuoso/Whitepapers/pdf/Virtuoso_SQL_to_RDF_Mapping.pdf.

Paper B

Modeling and Querying Spatial Data Warehouses on the Semantic Web

Nurefşan Gür, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi

The paper has been published in the
Proceedings of the 5th Joint International Semantic Technology Conference
Vol. 9544, pp. 3–22, 2015. DOI: 10.1007/978-3-319-31676-5_1

Abstract

The Semantic Web (SW) has drawn the attention of data enthusiasts, and also inspired the exploitation and design of multidimensional data warehouses, in an unconventional way. Traditional data warehouses (DW) operate over static data. However multidimensional (MD) data modeling approach can be dynamically extended by defining both the schema and instances of MD data as RDF graphs. The importance and applicability of MD data warehouses over RDF is widely studied yet none of the works support a spatially enhanced MD model on the SW. Spatial support in DWs is a desirable feature for enhanced analysis, since adding encoded spatial information of the data allows to query with spatial functions. In this paper we propose to empower the spatial dimension of data warehouses by adding spatial data types and topological relationships to the existing QB4OLAP vocabulary, which already supports the representation of the constructs of the MD models in RDF. With QB4SOLAP, spatial constructs of the MD models can be also published in RDF, which allows to implement spatial and metric analysis on spatial members along with OLAP operations. In our contribution, we describe a set of spatial OLAP (SOLAP) operations, demonstrate a spatially extended metamodel as, QB4SOLAP, and apply it on a use case scenario. Finally, we show how these SOLAP queries can be expressed in SPARQL.

1 Introduction

The evolution of the Semantic Web (SW) and its tools allow to employ complex analysis over multidimensional (MD) data models via On-Line Analytical Processing (OLAP) style queries. OLAP emerges when executing complex queries over data warehouses (DW) to support decision making. DWs store large volumes of data which are designed with MD modeling approach and usually perceived as *data cubes*. Cells of the cube represent the observation *facts* for analysis with a set of attributes called *measures* (*e.g.* a sales fact cube with measures of product quantity and prices). Facts are linked to *dimensions* which give contextual information (*e.g.* sales date, product, and location). Dimensions are perspectives which are used to analyze data, organized into *hierarchies* and *levels* that allow users to analyze and aggregate measures at different levels of detail. Levels have a set of *attributes* that describe the characteristics of the level members.

In traditional DWs, the "location" dimension is widely used as a conventional dimension which is represented in an alphanumeric manner with only nominal reference to the place names. This neither allow manipulating location-based data nor deriving topological relations among the hierarchy levels of the location dimension. This issue yields a demand for truly spatial DWs for better analysis purposes. Including encoded geometric information of the location data significantly improves the analysis process (*i.e.* proximity analysis of the locations) with comprehensive perspectives by revealing dynamic spatial hierarchy levels and new spatial members. The scope of this work is first focuses on enhancing the spatial characteristics of the cube members on the SW, and then describing and utilizing SOLAP operators for advanced analysis and decision making.

In our approach we consider enabling SOLAP capabilities directly over Resource Description Framework (RDF) data on the SW. Importance and applicability of performing OLAP operations directly over RDF data is studied in [1, 2]. To perform SOLAP over the SW consistently, an explicit and precise vocabulary is needed for the modeling process. The key concepts of spatial cube members need to be defined in advance to realize SOLAP operations since they employ spatial measures with spatial aggregate functions (*e.g.* union, buffer, and, convex-hull) and topological relations among spatial dimension and hierarchy level members (*e.g.* within, intersects, and, overlaps). Current state of the art RDF and OLAP technologies is limited to support conventional dimension schema and analysis along it's levels. Spatial dimension schema and SOLAP require an advanced specialized data model. As a first effort to overcome the limitations of modeling and querying spatial data warehouses on the Semantic Web we give our contributions in the following.

Contributions. We propose an extended metamodel solution that enables representation and RDF implementation of spatial DWs. We base our metamodel on the most recent QB4OLAP vocabulary and present an extension to support the spatial functions and spatial elements of the MD cubes. We discuss the notion of a SOLAP operator and observe it with examples, then we give the semantics of each SOLAP operator formally and finally, show how to implement them in SPARQL by using sub-queries and nested set of operators.

In the remainder of the paper, we first present the state of the art, in Sect. 2. As a prerequisite for our contribution in Sect. 3, we give the preliminary concepts and explain the structure of a SOLAP operator. Then, in Sect. 4 we define the semantics of MD data cube elements in RDF, present QB4SOLAP and formalize the SOLAP operators over MD data cube elements. We present a QB4SOLAP use case in Sect. 5 and then, we show how to write the defined SOLAP queries over this use case in SPARQL in Sect. 6. Finally, in Sect. 7, we conclude and remark to the future work directions.

2 State of the Art

DW and OLAP technologies have been proven a successful approach for analysis purposes on large volumes of data [3]. Aligning DW/OLAP technologies with RDF data makes external data sources available and brings up dynamic scenarios for analysis. The following studies are found concerning DW/OLAP with the SW.

DW/OLAP and Semantic Web: The potential of OLAP to analyze SW data is recognized in several approaches, thus MD modeling from ontologies is studied in the works of [4, 5]. However these approaches do not support standard querying of RDF data in SPARQL but require a MD or a relational database query engine, which limits the access to frequently updated RDF data. Kämpgen *et al.* propose an extended model [2] on top of RDF Data Cube Vocabulary (QB) [6] for interacting with statistical linked data via OLAP operations in SPARQL, but it has the limitations of the QB and thus cannot support full OLAP dimension structure with aggregate functions. It also has only limited support for complete MD data model members (*e.g.* hierarchies and levels). Etcheverry *et al.* introduce QB4OLAP [1] as an extended vocabulary of QB with a full MD metamodel, which supports OLAP operations directly over RDF data with SPARQL queries. However, none of these vocabularies and approaches support spatial DWs, unlike our proposal.

Spatial DW and OLAP: The constraint representation of spatial data has been focus in many fields from databases to AI [7]. Extending OLAP with spatial features has attracted the attention of data warehousing communities as well. Several conceptual models are proposed for representing spatial data in data

warehouses. Stefanovic *et al.* [8] investigates on constructing and materializing the spatial cubes in their proposed model. The MultiDim conceptual model is introduced by Malinowski and Zimányi [9] which copes with spatial features and extended in [10], to include complex geometric features (*i.e.* continuous fields), with a set of operations and MD calculus supporting spatial data types. Gómez *et al.* [11] propose an algebra and a very general framework for OLAP cube analysis on discrete and continuous spatial data. Even though spatial data warehousing is widely studied, it has not implemented yet on the Semantic Web.

Geospatial Semantic Web: The Open Geospatial Consortium – OGC pursue an important line of work for geospatial SW with GeoSPARQL [12] as a vocabulary to represent and query spatial data in RDF with an extension to SPARQL. Kyzirakos *et al.* presents a comprehensive survey in data models and query languages for linked geospatial data in [13], and propose a semantic geospatial data store - Strabon with an extensive query language – stSPARQL in [14], which is yet limited to a specific environment. LinkedGeoData is a significant contribution on interactively transforming OpenStreetMap[1] data to RDF data [15]. GeoKnow [16] is a more recent project with focus on linking geospatial data from heterogeneous sources.

The studies shows that, SW and RDF technologies evolve to give better functionality and standards for spatial data representation and querying. It is also argued above that spatial data is very much needed for DW/OLAP applications. However modeling and querying of spatial DWs on the SW is not addressed in any of the above papers. There are recent efforts on creating an Extract-Load-Transform (ETL) framework from semantic data warehouses [17] and publishing/converting open spatial data as Linked Open Data [18], which motivates modeling and querying spatial data warehouses on the Semantic Web. Spatial data requires specific treatment techniques, particular encoding, special functions and different manipulation methods, which should be considered during the design and modeling process. Current state of the art geospatial Semantic Web focuses on techniques for publishing, linking and querying spatial data however does not elaborate on analytical spatial queries for MD data. In order to address these issues we propose a generic and extensible metamodel based on the best practices of MD data publishing in RDF. Then we show how to create spatial analytical queries with SOLAP on MD data models. We base ourselves on existing works by extending the most recent version of the QB4OLAP vocabulary with spatial concepts. Furthermore, we introduce the new concept of SOLAP operators that navigate along spatial dynamic hierarchy levels and implement these analytical spatial queries in SPARQL.

[1] `http://www.openstreetmap.org`

3 Spatial and OLAP Operations

In this section we give define the spatial and spatial OLAP (SOLAP) operations.

3.1 Spatial Operations

In order to understand spatial operations, it is important to understand what is a *spatial object*. A spatial object is the data or information that identifies a real-world entity of geographic features, boundaries, places etc. Spatial objects can be represented in object/vector or image/raster mode. Database applications that can store spatial objects need to specify the spatial characteristics, encoded as specific information such as *geometry* data type which is the most common and supports planar or Euclidean (flat-earth) data. *Point, Line, and, Polygon* are the basic instantiable types of the geometry data type.

Geometries are associated with a *spatial reference system* (SRS) which describes the coordinate space in which the geometry is defined. There are several SRSs and each of them are identified with a *spatial reference system identifier* (SRID). The World Geodetic System (WGS) is the most well-known SRS and the latest version is called WGS84, which is also used in our use case.

Spatial data types have a set of operators that can function among applications. We grouped these operations into classes. Our classification is based on the common functionality of the operators. These classes are defined as follows:

Spatial Aggregation. The operators in the spatial aggregation, $\mathcal{S}_{agg}$ class aggregate two or more spatial objects. The result of these operators returns a new composite spatial object. Union, Intersection, Buffer, ConvexHull, and, MBR - Minimum Bounding Rectangle are example operators of this class.

Topological Relation. The operators in the topological relation, $\mathcal{T}_{rel}$ class are commonly contained in the RCC8[2] and DE-9DIM[3] models. Topological relations are standardized by OGC as Boolean operators which specify how two spatial objects are related to each other with a set of spatial predicates for example: Intersects, Disjoint, Equals, Overlaps, Contains, Within, Touches, Covers, CoveredBy, and, Crosses.

Numeric Operation. The operators in the class of numeric operation, $\mathcal{N}_{op}$ take one or more spatial objects and return a numeric value. Perimeter, Area, # of Interior Rings, Distance, Haversine Distance, Nearest Neighbor (NN), and # of Geometries are some of the example operators of this class.

[2]RCC8 – Region Connection Calculus describes regions in Euclidean space, or in a topological space by their possible relations to each other.

[3]DE-9DIM – Dimensionally Extended Nine-Intersection Model is a topological model that describes spatial relations of two geometries in two-dimensions.

3.2 SOLAP Operations

OLAP operations emerge when executing complex queries over multidimensional (MD) data models. OLAP operations let us interpret data from different perspectives at different levels of detail. *Spatially extended multidimensional models* incorporate spatial data during the analysis and decision making process by revealing new patterns, which are difficult to discover otherwise. In connection with our definition of MD models in the first paragraph of Sect. 1, hereafter we enhance and describe the spatially extended MD data cube elements.

A spatially extended MD model contains both conventional and spatial dimensions. A *spatial dimension* is a dimension which includes at least one *spatial hierarchy*. Dimensions usually have more than one level which are represented through hierarchies and there is always a unique top level *All* with just one member. A hierarchy is a *spatial hierarchy* if it has at least one *spatial level* in which the application should store the spatial characteristics of the data, which is captured by it is geometry and can be recorded in the *spatial attributes* of the level. A *spatial fact* is a fact that relates several dimensions in which, two or more are spatial. For example, consider a "Sales" *spatial fact* cube, which has "Customer" and "Supplier" (company) as *spatial dimensions* with a *spatial hierarchy* as "Geography" that expands into (hierarchic) *spatial levels*; "City $\rightarrow$ State $\rightarrow$ Country $\rightarrow$ Continent $\rightarrow$ *All*" from the customer and supplier's location. All these *spatial levels* record the spatial characteristics *i.e.* with a *spatial attribute* of a city (center) as "point" coordinates. Measures in the cube express additional and essential information for each MD data cell which is not exhibited through the dimensions and level attributes. Typically, spatially extended MD models have *spatial measures* which are represented by a geometry *i.e.* point, polygon, etc.

Spatial OLAP operates on spatially extended MD models. SOLAP enhances the analytical capabilities of OLAP with the spatial information of the cube members. The term SOLAP used in [19] and their similar works as a visual platform, which is designed to analyze huge volumes of geo-referenced data by providing visualization of pivot tables, graphical displays and interactive maps. We define the term SOLAP concisely as a platform (and query language) independent high-level concept, which is applicable on any spatial multidimensional data. We explain and exemplify in the following how SOLAP operators are interpreted.

Each operator in SOLAP should include at least one *spatial condition* by using the aforementioned operators from the spatial operation classes defined in Sect. 3.1. Spatial operations in SOLAP create a dynamic interpretation of the cube members as a *dynamic spatial hierarchy or level*. These interpretations allow new perspectives to analyze the spatial MD data which cannot be accessed in a traditional MD model. For instance, the classical OLAP opera-

tor *roll-up* aggregates measures along a hierarchy to obtain data at a coarser granularity. In the spatial dimension schema Fig. B.1, the (classical) roll-up relation, *Customer to City* is shown with black straight arrows. On the other hand, in SOLAP, a new "dynamic spatial hierarchy" is created on the fly to roll-up among spatial levels by a spatial condition (closest distance), which is given as *Customer to Closest-City (of the Supplier)*, shown with curved arrows in gray. The details of this operator in comparison with OLAP and SOLAP are given in the following example.

Example: Roll-up. The user wants to sum the total amount of the sales to customers up to the city level with the roll-up operator. The instance data for *Sales fact* is given in Tab. B.1 and shown on the map in Fig. B.2. The amount of the sales are shown in parentheses along with the quantities of the sold parcels (from supplier to customer). The arrows on the map, between the supplier and customer locations are used to represent the distance. The summarized data for sale instances (Tab. B.1) does not originally contain the records of the supplier – customer distance (as given in Tab. B.4) which can lead to increase in the storage space. If there are no sales to customers from the corresponding suppliers, a dash (–) is used in Tab. B.1. The syntax of the traditional roll-up operator is **ROLLUP(Sales, (Customer → City), SUM(SalesAmount))** which aggregates the *"total sales to customers up to city level"* (results in Tab. B.2). Alternatively, the user may like to view the "*total sales to customers by city of the closest suppliers*", in which some customers can be closer to their suppliers from other cities, as emphasized in Tab. B.4. This query is possible with traditional OLAP, if only Tab. B.4 is recorded in the base data which requires extra storage space. For a better support and flexibility we define a spatial roll-up operator that aggregates the total sales along

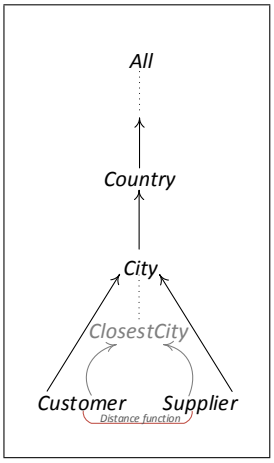


Fig. B.1: S-Dim. Schema

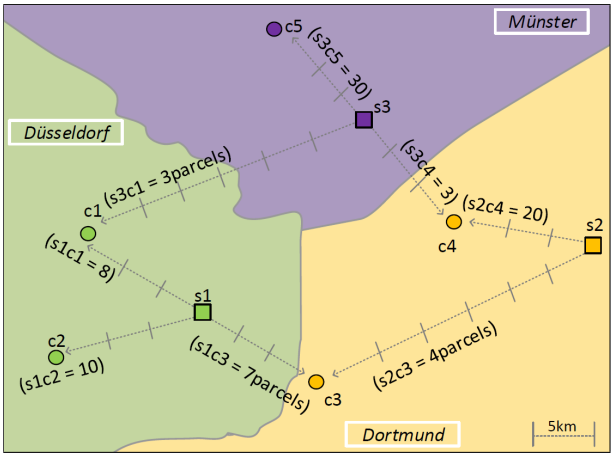


Fig. B.2: Example Map of Sales (Instance) Data

the dynamic spatial hierarchy, which is created based on a spatial condition (in this example, distance between customer and supplier locations). The syntax for s-roll-up is; **S-ROLLUP(Sales,[CLOSEST(Customer,Supplier)] → City',SUM(SalesAmount))**. The spatial condition transforms the customer→ city hierarchy as a dynamic spatial hierarchy, which depends on the proximity of the suppliers that is calculated during runtime. The user has the flexibility to make analyses with different spatial operations in SOLAP. Spatial extensions of common OLAP operators (roll-up, drill-down, slice, and dice) are formally defined in Sect. 4.3.

City	Customer	Supplier Sales			Total Sales
		s1	s2	s3	
Düsseldorf	c1	8pcs.	–	3pcs.	11pcs.
	c2	10pcs.	–	–	10pcs.
Dortmund	c3	7pcs.	4pcs.	–	11pcs.
	c4	–	20pcs.	3pcs.	23pcs.
Münster	c5	–	–	30pcs.	30pcs.

Table B.1: Sample (Instance) Data for Sales

City	Sales
Düsseldorf	21pcs.
Dortmund	34pcs.
Münster	30pcs.

Table B.2: Roll-up

City	Sales
Düsseldorf	25pcs.
Dortmund	20pcs.
Münster	33pcs.

Table B.3: S-Roll-up

	Sup. City	Düsseldorf	Dortmund	Münster
Cust. City	Sup. / Cust.	s1	s2	s3
Düsseldorf	c1	15 km.s	45km.s	30 km.s
	c2	15 km.s	60 km.s	60 km.s
Dortmund	c3	*15 km.s*	30 km.s	45 km.s
	c4	45 km.s	15 km.s	*15 km.s*
Münster	c5	60 km.s	45 km.s	15 km.s

Table B.4: Customer to Supplier Distance (km.s)

4 Semantics of Spatial MD Data and OLAP Operations

In this section, we first present our approach on how to support spatial MD data in RDF by using the QB4SOLAP vocabulary. Afterwards, we define the general semantics of each SOLAP operator to be implemented in SPARQL as a proof of concept to the QB4SOLAP metamodel. The concepts introduced in this metamodel are an extension to the most recent QB4OLAP vocabulary [1].

Figure B.3 shows the proposed and extended QB4OLAP vocabulary for the *cube schema* RDF triples. Capitalized terms in the figure represent RDF classes and non-capitalized terms represent RDF properties. Classes in external vocabularies are depicted in light gray background and font. RDF Cube (QB), QB4OLAP, QB4SOLAP classes are shown with white, light gray, dark gray backgrounds, respectively. Original QB terms are prefixed with `qb:`. QB4OLAP and QB4SOLAP classes and properties are prefixed with `qb4o:` and `qb4so:`. In order to represent spatial classes and properties an external prefix from OGC, `geo:` is used in the metamodel. Since QB4OLAP and QB4SOLAP are RDF-based multidimensional model schemas, we first define formally what an RDF triple is, and then discuss the basics of describing MD data using QB4OLAP and spatially enhanced MD data in QB4SOLAP.

An *RDF triple* t consists of three components; s is the subject, p is the predicate, and o is the object, which is defined as: *triple (s,p,o)* $\in t = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}i)$ where the set of *IRIs* is $\mathcal{I}$, the set of *blank nodes* is $\mathcal{B}$, and the set of *literals* is $\mathcal{L}i$. Given an MD element x of the cube schema $\mathcal{CS}$, $\mathcal{CS}(x) \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}i)$ returns a set of triples $\mathcal{T}$, with IRIs, blank nodes and literals. The notation of the triples in the following definitions is given as $(x$ `rdf:type ex:SomeProperty`$)$. If the concepts are defined with a set of triples, after the first triple, we use a semicolon "`;`" to link predicates (p) and objects (o) with the subject (s) concept x. The blank nodes $\mathcal{B}$ are expressed as `_:` and nesting unlabeled blank nodes are abbreviated in square brackets "`[]`".

4.1 Defining MD Data in QB4OLAP

In order to explain the spatially enhanced MD models we first describe MD elements described in Sect. 1 with the RDF formalization in QB4OLAP vocabulary.

Cube Schema. A data structure definition (DSD) specifies the schema of a data set (*i.e.*, a cube, which is an instance of the class `qb:DataSet`). The DSD can be shared among different data sets. The DSD of a data set represents dimensions, levels, measures, and attributes with component properties. The DSD is defined through a conceptual MD cube schema $\mathcal{CS}$, which has a

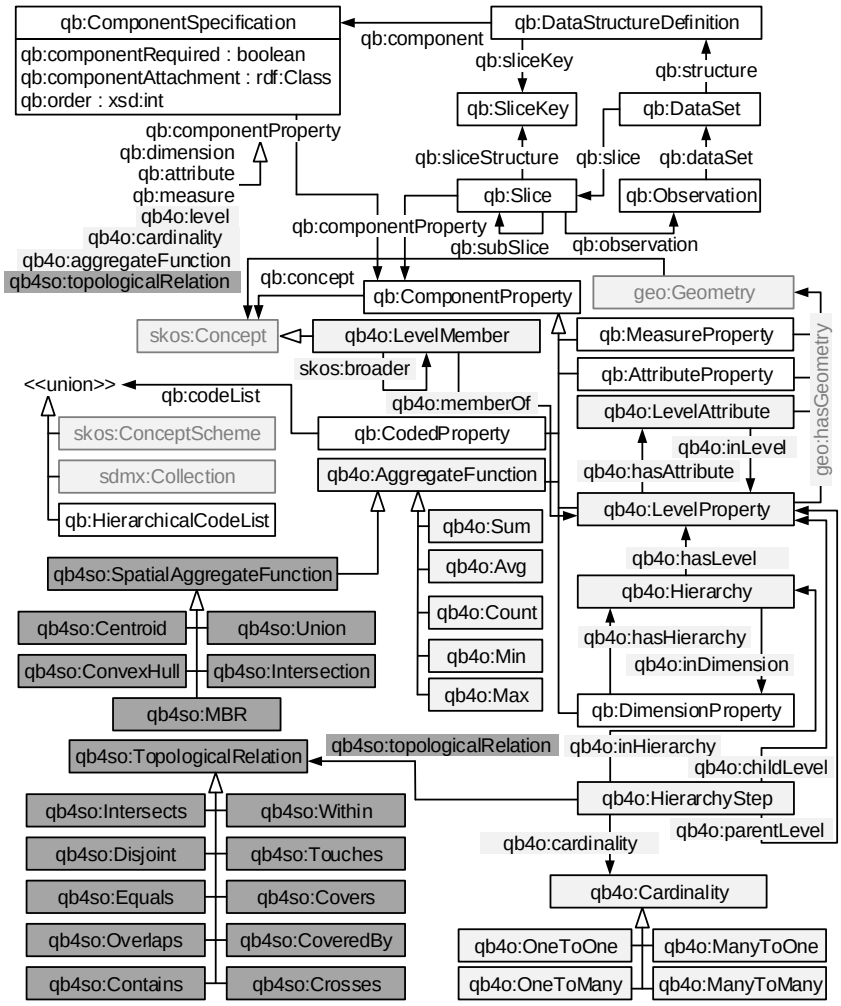


Fig. B.3: QB4SOLAP Vocabulary Meta-Model

set of dimension types $\mathcal{D}$, a set of measures $\mathcal{M}$ and, with a fact type $\mathcal{F}$ as $\mathcal{CS} = (\mathcal{D}, \mathcal{M}, \mathcal{F})$. For example, a cube schema $\mathcal{CS}$ can be used to define a physical structure of a company's sales data to be represented as a MD data cube. We define the cube schema elements in the following definitions.

Attributes. An attribute $a \in \mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ has a domain $\langle a : dom \rangle$ in the cube schema $\mathcal{CS}$ with a set of triples $t_a \in \mathcal{T}$ where t_a is encoded as $(a$ `rdf:type qb:AttributeProperty; rdfs:domain xsd:Schema`$)$

The domain of the attribute is given with the property `rdfs:domain`[4] from the corresponding schema and `rdfs:range` defines what values the property can take *i.e.*; *integer*, *decimal*, etc. from the given, $xsd:Schema$ elements[5]. Attributes are the finest granular elements of the cube, which exists in levels to describe the characteristics of level members *e.g.*, customer level attributes could be as; name, id, address, etc.

Levels. A level $l \in \mathcal{L} = \{l_1, l_2, \ldots, l_n\}$ consists of a set of attributes $\mathcal{A}_l$, which is defined by a schema $l(a_1 : dom_1, \ldots, a_n : dom_n)$, where l is the level and each attribute a is defined over the domain dom. For each level $l \in \mathcal{L}$ in the cube schema $\mathcal{CS}$, there is a set of triples $t_l \in \mathcal{T}$ which is encoded as (l `rdf:type qb4o:LevelProperty; qb4o:hasAttribute` a). Relevant levels for customer data include; customer level, city level, country level, etc.

Hierarchies. A hierarchy $h \in \mathcal{H} = \{h_1, h_2, \ldots, h_n\}$ in the cube schema $\mathcal{CS}$, is defined with a set of triples $t_h \in \mathcal{T}$, and encoded as (h `rdf:type qb4o:HierarchyProperty; qb4o:hasLevel` l; `qb4o:inDimension` $\mathcal{D}$).

Each hierarchy $h \in \mathcal{H}$ is defined as $h = (\mathcal{L}_h, \mathcal{R}_h)$; with a set of $\mathcal{L}_h$ (hierarchy) levels, which is a subset of the set $\mathcal{L}_d$ levels of the dimension $\mathcal{D}$ where $\mathcal{L}_h \subseteq \mathcal{L}_d \in \mathcal{D}$. $\mathcal{L}_d$ contains the initial base level of the dimension in addition to hierarchy levels L_h. For example, customer–location hierarchy can be defined by the levels; customer, city, country, etc. where customer is the base level and contained only in $\mathcal{L}_d$.

Due to the nature of the hierarchies, a hierarchy entails a roll-up relation $\mathcal{R}_h$ between its levels, $\mathcal{R}_h = (\mathcal{L}_c, \mathcal{L}_p, card)$ where $\mathcal{L}_c$ and $\mathcal{L}_p$ are respectively child and parent levels, where the lower level is called child and higher level is called parent. Cardinality $card \in \{1-1, 1-n, n-1, n-n\}$ describes the minimum and maximum number of members in one level that can be related to a member in another level, *e.g.*, $\mathcal{R}_h = (city, country, many-to-one)$ shows that the roll-up relation between the child level city to parent level country is many-to-one, which means that each country can have many cities. In order to represent cardinalities between the child and the parent levels, blank nodes are created as hierarchy steps, $:\_h_{hs} \in \mathcal{B}$. Hierarchy steps relate the levels of the hierarchy from a bottom (child) level to an upper (parent) level, which is defined with a set of triples $t_{hs} \in \mathcal{T}$ and encoded as ($:\_h_{hs}$ `rdf:type qb4o:HierarchyStep; qb4o:childLevel` lh_c; `qb4o:parentLevel` lh_p; `qb4o:cardinality` $card$) where $lh_c \in \mathcal{L}_c$, $lh_p \in \mathcal{L}_p$ and $card \in \{1-1, 1-n, n-1, n-n\}$.

Dimensions. An n-dimensional cube schema has a set of dimensions $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$. And each $d \in \mathcal{D}$ is defined as a tuple $d = (\mathcal{L}, \mathcal{H})$; with a set of $\mathcal{L}_d$ levels, organized into $\mathcal{H}_d$ hierarchies. Dimensions, inherently have all the levels from the hierarchies they have, and an initial base level. For each di-

[4]RDF Schema `http://www.w3.org/TR/rdf-schema/`

[5]XML Schema `http://www.w3.org/TR/xmlschema11-1/`

mension $d \in \mathcal{D}$, in the cube schema $\mathcal{CS}$, there is a set of triples $t_d \in \mathcal{T}$, which is encoded as (d `rdf:type qb:DimensionProperty; qb4o:hasHierarchy` h). For example, customerDim is a dimension with a location and a customer type hierarchy where location expands to levels of customer's location (e.g., city, country, etc.) and customer type expands to levels of customer's type (e.g.,profession, branch, etc.)

Measures. A measure $m \in \mathcal{M} = \{m_1, m_2, \ldots, m_n\}$, is a property, which is associated to the facts. Measures are given in the cube schema $\mathcal{CS}$ with a set of triples $t_m \in \mathcal{T}$, which is encoded as (m `rdf:type qb:MeasureProperty; rdfs:subPropertyOf sdmx-measure:obsValue; rdfs:domain xsd:Schema`).

Measures are defined with a sub-property from the Statistical Data and Metadata Exchange - (*sdmx*) definitions, `sdmx-measure:obsValue` which is the value of a particular variable for a particular observation[6]. Similarly to the attributes `rdfs:domain` specifies the schema of the measure property and, `rdfs:range` defines what values the property can take *i.e.; integer, decimal, etc.* in the instances. For example, quantity and price are measures of a fact (e.g., sales) where the instance values can be given respectively, in the form: "13"^^xsd:positiveInteger and "42.40"^^xsd:decimal. Measures are associated with observations (facts) and related to dimension levels in the DSD as explained in the following.

Facts. A fact $f \in \mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is related to values of dimensions and measures. The relation is described in *components* in the schema level of the facts cube definition, by a set of triples $t_f \in \mathcal{T}$, which is encoded as ($\mathcal{F}$ `rdf:type qb:DataStructureDefinition; qb:component[qb4o:level` l`; qb4o:cardinality` *card*`]; qb:component[qb:measure` m`; qb4o:aggregateFunction` *BIF*`]`). Cardinality, $card \in \{1-1, 1-n, n-1, n-n\}$ represents the cardinality of the relationship between facts and level members. The specification of the aggregate functions for measures is required in the definition of the cube schema. Standard way of representing typical aggregate functions is defined by QB4OLAP namely built-in functions such as; $BIF \in \{Sum, Avg, Count, Min, Max\}$. For example, a fact schema $\mathcal{F}$ can be sales of a company which has associated dimensions and measures defined as components respectively e.g. product and price.

Finally, the facts $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ are given on the instance level where each fact f has a unique IRI $\mathcal{I}$, which are observations. This is encoded as (f `rdf:type qb:Observation`). An example of a fact instance f with it's relation to measure values and dimension levels is a "sale" transacted to customer "John" (value of the dimension level), for a product "chocalate"(value of another dimension level), which has a unit price of "29.99" (value of a measure) euros, and quantity of "20" (value of another measure) boxes. Cardinality of the dimension level customer and fact member is many–to–one where several

[6] `http://sdmx.org/`

sales can be transacted to the same customer (i.e. John). Specification of the aggregate function for measure unit price is “average" while quantity can be specified as “sum".

We gave the cube schema $\mathcal{CS} = (\mathcal{D}, \mathcal{M}, \mathcal{F})$ members above where dimensions $d \in \mathcal{D}$ are defined as a tuple of dimension levels $\mathcal{L}_d$ and hierarchies $\mathcal{H}$, $d = (\mathcal{L}_d, \mathcal{H})$, and a hierarchy $h \in \mathcal{H}$ is defined with hierarchy levels $\mathcal{L}_h$ such that $h = (\mathcal{L}_h)$, where $\mathcal{L}_h \subseteq \mathcal{L}_d$, and a level l contains attributes $\mathcal{A}_l$ as $l = (\mathcal{A}_l)$.

4.2 Defining Spatially Enhanced MD Data in QB4SOLAP

QB4SOLAP adds a new concept to the metamodel, which is `geo:Geometry` class from OGC schemas[7]. We define the QB4SOLAP extension to the cube schema in the description of the following spatial MD data elements, which are explained in Sect. 3.2.

Spatial Attributes. Each attribute is defined over a domain (Sect. 4.1). Every attribute with geometry domain ($a : dom_g \in \mathcal{A}$) is a member of `geo:Geometry` class and they are called spatial attributes a_s, which are defined in the cube schema $\mathcal{CS}$ by a set of triples $t_{ag} \in \mathcal{T}$ and encoded as (a_s `rdf:type qb:AttributeProperty; rdfs:domain geo:Geometry`). The type (point, polygon, line, *etc.*) of the each spatial attribute is assigned with `rdfs:range` predicate in the instances. For example, a spatial attribute can be the “capital city" of a country which is represented through a point geometry.

Spatial Levels. Each spatial level $l_s \in \mathcal{L}$ is defined with a set of triples $t_{ls} \in \mathcal{T}$ in the cube schema $\mathcal{CS}$, and encoded as (l_s `rdf:type qb4o:LevelProperty; qb4o:hasAttribute` a, a_s`; geo:hasGeometry geo:Geometry`). Spatial levels must be a member of `geo:Geometry` class and might have spatial attributes. For example country level is a spatial level which has a polygon geometry and might also record geometry of the capital city in the level attributes as a point type.

Spatial Hierarchies. Each hierarchy $h_s \in \mathcal{H}$ is spatial , if it relates two or more spatial levels l_s. Spatial hierarchy step defines the relation between the spatial levels with a roll-up relation as in conventional hierarchy steps (Sect. 4.1). QB4SOLAP introduces topological relations, $\mathcal{T}_{rel}$ (Sect. 3.1) besides cardinalities in the roll-up relation which is encoded as $\mathcal{R} = (\mathcal{L}_c, \mathcal{L}_p, card, \mathcal{T}_{rel})$ for the spatial hierarchy steps.

Let $t_{shs} \in \mathcal{T}$ a set of triples to represent a hierarchy step for spatial levels in hierarchies, which is given with a blank node $:\_sh_{hs} \in \mathcal{B}$ and encoded as ($:\_sh_{hs}$ `rdf:type qb4o:HierarchyStep; qb4o:childLevel` slh_{ci}`; qb4o:parentLevel` slh_{pi}`; qb4o:cardinality` $card$`; qb4so:hasTopologicalRelation` $\mathcal{T}_{rel}$) where $slh_{ci} \in \mathcal{L}_c$, $slh_{pi} \in \mathcal{L}_p$. For example, a spatial hierarchy

[7]OGC Schemas `http://schemas.opengis.net/`

is “geography" which should have spatial levels (e.g. customer, city, country, and continent) with the roll-up relation $\mathcal{R}_h = (city, country, many-to-one, within)$, which also specifies that child level city is “within" the parent level country, in addition to the hierarchy steps from Sect. 4.1.

Spatial Dimensions. Dimensions are identified as spatial if only they have at least one spatial hierarchy. More than one dimension can share the same spatial hierarchy and the spatial levels, which belongs to that hierarchy. QB4SOLAP uses the same schema definitions of the dimensions as in Sect. 4.1. For example, a spatial dimension is customer dimension, which has a spatial hierarchy geography.

Spatial Measures. Each spatial measure $m_s \in \mathcal{M}$ is defined in the cube schema $\mathcal{CS}$ by a set of triples $t_{ms} \in \mathcal{T}$ and encoded as (m_s `rdf:type qb:MeasureProperty; rdfs:subPropertyOf sdmx-measure:obsValue; rdfs:domain geo:Geometry`). The class of the numeric value is given with the property `rdfs:domain` and `rdfs:range` assigns the values from the class `geo:Geometry`, *i.e.*, *point*, *polygon*, *etc.* at the instance level.

Spatial measures are represented by a geometry thus they use a different schema than conventional (numeric) measures. The schemas for spatial measures have common geometry serialization standards[8] that are used in OGC schemas. For example a spatial measure is coordinates of an accident location, which is given as a point geometry type and associated to an observation fact of accidents.

Spatial Facts. Spatial facts $\mathcal{F}_s$ relates several dimensions of which two or more are spatial. If there are several spatial dimension levels (l_s), related to the fact, topological relations $\mathcal{T}_{rel}$ (Sect. 3.1) between the spatial members of the fact instance may be required which is not necessarily imposed for all the spatial fact cubes. Ideally a spatial fact cube has spatial measures (m_s), as its members which makes it possible to aggregate along spatial measures with the spatial aggregation functions $\mathcal{S}_{agg}$ (Sect. 3.1). Representation of a complete spatial fact cube at the schema level in RDF is given by a set of triples $t_{fs} \in \mathcal{T}$, and encoded as ($\mathcal{F}_s$ `a qb:DataStructureDefinition; qb:component [qb4o:level` l_s`; rdfs:subPropertyOf sdmx-dimension:refArea; qb4o:cardinality` *card*`; qb4so:TopologicalRelation` $\mathcal{T}_{rel}$`]; qb:component [qb:measure` m_s`, sdmx-measure:obsValue; qb4o:aggregateFunction` BIF'`]`). QB4SOLAP extends the built-in functions of QB4OLAP with spatial aggregation functions as $BIF' = BIF \cup \mathcal{S}_{agg}$, which is added with a class `qb4so:Spatial AggregateFunction` to the metamodel in Fig. B.3. An example of a spatial fact instance f_s with it's relation to measure values and dimension levels is a traffic “accident" incident occurred on a highway “E–45"

[8]The Well Known Text (WKT) serialization aligns the geometry types with ISO 19125 Simple Features [ISO 19125-1], and the GML serialization aligns the geometry types with [ISO 19107] Spatial Schema.

(value of the highway spatial dimension level) with coordinate points of the location “57.013, 9.939" (value of the location spatial measure). Cardinality of the dimension level highway and fact member is many–to–one where several accidents might take place in the same highway. Specification of the spatial aggregate function for spatial measure location (coordinate points) can be specified as “convex hull" area of the accident locations.

4.3 SOLAP Operators

The proposed vocabulary QB4SOLAP allows publishing spatially enhanced multidimensional RDF data which allows us to query with SOLAP operations. Subqueries and aggregation functions in SPARQL 1.1[9] make it easily possible to operate with OLAP queries on multidimensional RDF data. Moreover, spatially enhanced RDF stores, provide functions to an extent for querying with topological relations and spatial numeric operations. In the following, we define common OLAP operators with spatial conditions in order to formalize spatial OLAP query classes. Spatial conditions can be selected from a range of operation classes that can be applied on spatial data types (Sect. 3.1). Let $\mathcal{S}$ be any spatial operation where $\mathcal{S} = (\mathcal{S}_{agg} \cup \mathcal{T}_{rel} \cup \mathcal{N}_{op})$ to represent a spatial condition in a SOLAP operation. The following OLAP operators are given with a spatial extension to the well-known OLAP operators defined over cubes based in Cube Algebra operators [20].

S–Roll–up. Given a cube $\mathcal{C}$, a dimension $d \in \mathcal{C}$, and an upper dimension level $l_u \in d$, such that $l \langle a : dom_g \rangle \rightarrow^* l_u$, where $l \langle a : dom_g \rangle$ represents the level in dimension d with attributes (a_s) whose domain is a geometry type. Let $\mathcal{R}_s$ be the spatial roll-up relation which comprises $\mathcal{S}$ and traditional roll-up relation $\mathcal{R}$ such that $\mathcal{R}_s = \mathcal{S}(d, l \langle a : dom_g \rangle) \cup \mathcal{R}(\mathcal{C}, d, l_u) \rightarrow \mathcal{C}'$.

Initially, in the semantics of *S–Roll–up* above, spatial constraint $\mathcal{S}$ is applied over a dimension d on the spatial attributes a_s along levels l. As a result of the roll–up relation $\mathcal{R}$, the measures are aggregated up to level l_u along d which returns a new cube $\mathcal{C}'$. Note that applying $\mathcal{S}$, on spatial level attributes a_s of dimension $\mathcal{D}$, operates on the hierarchy step $l \rightarrow l_u$ with a dynamic spatial hierarchy (Ref. Sect. 3.2). For example, the query “total sales to customers by city of the closest suppliers" implies a S-Roll-up operator.

S–Drill–down. Analogously, *S–Drill–down* is an inverse operation of *S–Roll–up*, which disaggregates previously summarized data down to a child level. For example, the query “average sales of the employees from the biggest city in its country" implies a S-Drill-down operator by disaggregating data from (parent) country level to (child) city level by imposing also a spatial condition (area from $\mathcal{N}_{op}$ to choose the biggest city) .

S–Slice. Given a cube $\mathcal{C}$ with n dimensions $\mathcal{D} = \{d_1, d_2, \ldots, d_n\} \in \mathcal{C}$, let $\mathcal{S}'$

[9] http://www.w3.org/TR/sparql11-query/

be the traditional slice operator which removes a dimension d from the cube $\mathcal{C}$. And let $\mathcal{S}_s$ be the spatial slice operator, which comprises $\mathcal{S}$, the spatial function to fix a single value in the level $\mathcal{L} = \{l_1, l_2, \ldots, l_n\} \in d$ defined as follows; $\mathcal{S}_s = \mathcal{S}'(\mathcal{C}, d) \cup \mathcal{S}(d, l\,\langle a : dom_g \rangle) \rightarrow \mathcal{C}'$.

Note that the spatial function is applied on the spatial attributes of the selected level, measures are aggregated along dimension d up to level All. The result returns a new cube $\mathcal{C}'$ with $n-1$ dimensions $\mathcal{D}' = \{d_1, d_2, \ldots, d_{n-1}\} \in \mathcal{C}'$. For example, the query "total sales to the customers located in the city within a 10 km. buffer area from a given point" implies a S-Slice operator, which dynamically defines the city level by (fixing) a specified buffer area around a given custom point in the city.

S–Dice. Dice operation is analogous to relational algebra - *R selection*; $\sigma_\phi(R)$, instead the argument is a cube $\mathcal{C}$; $\sigma_\phi(\mathcal{C})$. In SOLAP dice is not a select operation rather a nested "select" and a "spatial filter" operation. S-Dice $\mathcal{D}_s$ keeps the cells of a cube $\mathcal{C}$ that satisfy a spatial Boolean $\mathcal{S}(\phi)$ condition over spatial dimension levels, attributes and measures which is defined as; $\mathcal{D}_s = (\mathcal{C}, \mathcal{S}(\phi)) \rightarrow \mathcal{C}'$ where $\mathcal{S}(\phi) = \mathcal{S}(\sigma_{a\phi b}(\mathcal{C})) \vee \mathcal{S}(\sigma_{a\phi v}(\mathcal{C}))$ and a, b are spatial levels (l_s), geometry attributes $(a : dom_g)$ or measures (m, m_s) while v is a constant value and the result returns a sub-cube $\mathcal{C}' \subset \mathcal{C}$. For example, the query "total sales to the customers which are located less than 5 km from their city" implies a S-Dice operator.

In this paper, we focus on direct querying of single data cubes. The integration of several cubes through *S-Drill-across* or set-oriented operations such as *Union, Intersection, and Difference* [20] is out of scope and remained as future work. The actual use of these query classes in SPARQL with the instance data is given in Section 6.

5 Use Case Scenario: GeoNorthwind Data Warehouse

Figure B.4 consists of the conceptual schema of the The GeoNorthwind DW use case. GeoNorthwind DW has synthetic data about companies and their sales, however it is well suited for representing MD data modeling concepts due to its rich dimensions and hierarchies. It is a good proof of concept use case to show how to implement spatial data cube concepts on the SW. We show next how to express the conceptual schema of GeoNorthwind in QB4SOLAP.

In the use case, measures are given in the Sales cube. All measures are conventional. The members of the GeoNorthwind DW are given with `gnw:`

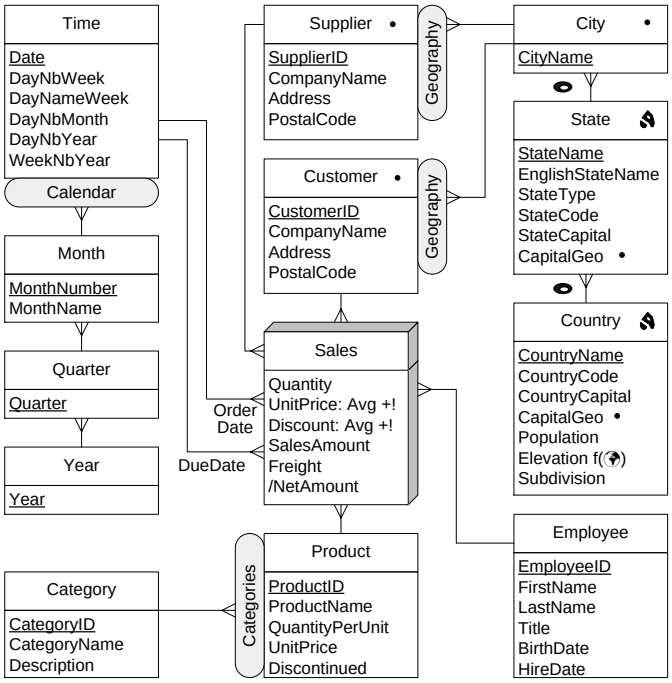


Fig. B.4: Conceptual MD Schema of the GeoNorthwind DW

prefix. The underlying syntax for RDF representation is given in Turtle[10] syntax in the boxes. An example of a measure in the cube schema is given in the following as defined in Sect. 4.1.

```
gnw:quantity a rdf:Property, qb:MeasureProperty;
rdfs:subPropertyOf sdmx-measure:obsValue;rdfs:range xsd:integer.
```

In the following, a spatial attribute of a spatial level `gnw:state` is given along with the level and attribute properties. Spatial level has a geometry as `gnw:statePolygon` independently having a spatial attribute `gnw:capitalGeo`. Each spatial attribute in the schema is defined separately by using common RDF and standard spatial schemas[11] to represent their domain and data type as described in Sect. 4.2.

```
gnw:state a qb4o:LevelProperty; qb4o:hasAttribute gnw:stateName,
   gnw:stateType, gnw:stateCapital, gnw:capitalGeo;
```

[10] `http://www.w3.org/TR/turtle/`

[11] For our tests we used Virtuoso Universal Server and `virtrdf:Geometry` is a special RDF typed literal which is used for geometry objects in Virtuoso. Normally, WGS84 (EPSG:4326) is the SRID of any such geometry.

```
   geo:hasGeometry gnw:statePolygon.
gnw:captialGeo a  qb:AttributeProperty;
  rdfs:domain geo:Geometry; rdfs:range geo:Point, geo:wktLiteral,
  virtrdf:Geometry.
```

In the next listing, an example of a spatial dimension from the use case data is `gnw:customerDim`, which is given with its spatial hieachy `gnw:geography` (Sect. 4.1, 4.2). The spatial hierarchy is organized into levels (*i.e.* city, state, country *etc.*) where `qb4o:hasLevel` predicate indicates the levels that compose the hierarchy. Each hierarchy in dimensions is represented with `qb4o:inDimension` predicate, referring to the dimension(s) it belongs to. The levels given in the dimension hierarchy are all spatial, and the sample representation of a spatial level is given above.

```
gnw:customerDim a rdf:Property, qb:DimensionProperty;
   qb4o:hasHierarchy gnw:geography.
gnw:geography a qb4o:HierarchyProperty; qb4o:hasLevel gnw:city,
   gnw:state, gnw:region, gnw:country, gnw:continent;
   qb4o:inDimension gnw:customerDim, gnw:supplierDim.
```

Each hierarchy step is added to the schema as a blank node (`_:hsi`) by `qb4o:HierarchyStep` property, in which the cardinality and topological relationships are represented in between the child and parent levels as follows;

```
_:hs1 a qb4o:HierarchyStep; qb4o:inHierarchy gnw:geography;
   qb4o:childLevel gnw:customer, gnw:supplier;
   qb4o:parentLevel gnw:city; qb4o:cardinality qb4o:ManyToOne;
   qb4so:hasTopologicalRelation qb4so:Within.
```

The components of the facts are described at the schema level in the cube definition. The dimension level for `gnw:customer` is given with `sdmx-dimenson:refArea` property, which indicates the spatial characteristic of the dimension. Measures require the specification of the aggregate functions in the cube definition. As there are only numeric measures in the use case data, aggregate function for the sample measure `gnw:quantity` is given as `qb4o:sum`. The general overview of the cube schema $\mathcal{CS}$ which is given with the related components as follows:

```
### Cube definition ###
gnw:GeoNorthwind rdf:type qb:DataStructureDefinition;
### Lowest level for each dimension in the cube ###
qb:component [qb4o:level gnw:customer, sdmx-dimension:refArea;
qb4o:cardinality qb4o:ManyToOne].
### Measures in the Cube ###
qb:component [qb:measure gnw:quantity; qb4o:aggregateFunction qb4o:sum].
```

A spatial fact cube may contain spatial measure components besides spatial dimension according to QB4SOLAP. The implementation scope of this work

covers only spatial facts, with spatial dimension and numerical measure components.

6 Querying the GeoNorthwind DW in SPARQL

We show next how some of the spatial OLAP queries from Sect. 4.3 can be expressed in SPARQL[12].

Query 1 (S-Roll-Up): Total sales to customers by city of the closest suppliers.

```
SELECT ?city (SUM(?sales) AS ?totalSales)
WHERE {?o a qb:Observation; gnw:customerID ?cust;
        gnw:supplierID ?sup; gnw:salesAmount ?sales.
        ?cust qb4o:inLevel gnw:customer;gnw:customerGeo ?custGeo;
        gnw:customerName ?custName; skos:broader ?city .
        ?city qb4o:inLevel gnw:city.?sup gnw:supplierGeo ?supGeo.
#Inner Select:Distance to the closest supplier of the customer
  {SELECT ?cust1 (MIN(?distance) AS ?minDistance)
  WHERE{?o a qb:Observation; gnw:customerID ?cust1;
        gnw:supplierID ?sup1. ?sup1 gnw:supplierGeo ?sup1Geo.
        ?cust1 gnw:customerGeo ?cust1Geo .
  BIND (bif:st_distance( ?cust1Geo, ?sup1Geo ) AS ?distance)}
  GROUP BY ?cust1 }
  FILTER (?cust = ?cust1 && bif:st_distance(?custGeo, ?supGeo)=
  ?minDistance)} GROUP BY ?city ORDER BY ?totalSales
```

The query above shows the spatial roll-up operation example from Sect. 3.2 with the actual use case data. We have explained the semantics of s-roll-up operator in Def. 11. The inner select verifies the spatial condition in order to find the closest distance to suppliers from the customers. The outer select prepares the traditional roll up of the total sales from customer (child) level to the city (parent) level. Filter on customer and supplier distance creates the aforementioned dynamic spatial hierarchy based on the proximity of the suppliers.

Query 2 (S-Slice): Total sales to the customers located in the city within a 10 km. buffer area from a given point.

```
SELECT ?custName ?cityName (SUM(?sales) AS ?totalSales)
WHERE {?o rdf:type qb:Observation; gnw:customerID ?cust;
   gnw:salesAmount ?sales. ?cust gnw:customerName ?custName;
   skos:broader ?city. ?city gnw:cityGeo ?cityGeo;
        gnw:cityName ?cityName.
FILTER(bif:st_within(?cityGeo, bif:st_point(2.3522,48.856),10))}
GROUP BY ?custName ?cityName ORDER BY ?custName
```

The semantics of the above (s-slice) operator is given in Def. 13. Traditional slice operator removes a dimension, by fixing a single value in a level of dimension with a given fixed value (*i.e.* CityName = “Paris"). On the

[12]SPARQL endpoint is available at: http://extbi.ulb.ac.be:8890/sparql.

other hand, s-slice dynamically defines the city level, by a specified buffer area around a given custom point in the city. Thus, s-slice removes the dimension customer and its instance in city Paris, but only the customer instances within 10 km. buffer area of the desired location. The project content with corresponding data sets and full query examples are available at: http://extbi.cs.aau.dk/QB4SOLAP/index.php.

7 Conclusion and Future Work

In this paper, we studied the modeling issues of spatially enhanced MD data cubes in RDF, defined the concept of SOLAP operators and implemented them in SPARQL. We showed that in order to model spatial DWs on the SW, an extended representation of MD cube elements was required. We based our representation on the most recent QB4OLAP vocabulary and make it viable for spatially enhanced MD data models through the new QB4SOLAP metamodel. This allows users to publish spatial MD data in RDF format. Then, we define well-known OLAP operations on data cubes with spatial conditions, in order to introduce spatial OLAP query classes and formally define their semantics. Subsequently, we present a use case and implement real-world SOLAP queries in SPARQL, to validate our approach.

Future work will be conducted in two areas: 1) defining complete formal techniques and algorithms for generating SOLAP queries in SPARQL based on a high-level MD Cube Algebra as in [20], and extending the coverage of SOLAP operations over multiple RDF cubes in SPARQL, *i.e.*, to support *S-Drill-Across*; 2) implement our QB4SOLAP approach on a more complex case study with spatial measures and facts which can support spatial aggregation (*S-Aggregation*) operator over measures with geometries. In order to support this *S-Aggregation* operator in SPARQL we will also investigate on creating user-defined SPARQL functions.

References

[1] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web using QB4OLAP," in *Data Warehousing and Knowledge Discovery (DaWaK'14)*, vol. 8646. Springer, 2014, pp. 45–56, https://dx.doi.org/10.1007/978-3-319-10160-6_5.

[2] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *The Semantic Web: ESWC 2012 Satellite Events*, vol. 7540. Springer, 2012, pp. 87–101, https://dx.doi.org/10.1007/978-3-662-46641-4_7.

[3] A. Abelló, O. Romero, T. Pedersen, R. Berlanga Llavori, V. Nebot, M. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, no. 2, pp. 571–588, 2014, https://doi.org/10.1109/TKDE.2014.2330822.

[4] C. Diamantini and D. Potena, "Semantic Enrichment of Strategic Datacubes," in *Proceedings of the 11th International Workshop on Datawarehousing and OLAP*. ACM, 2008, pp. 81–88.

[5] V. Nebot, R. Berlanga, J. M. Pérez, M. J. Aramburu, and T. B. Pedersen, "Multidimensional Integrated Ontologies: A Framework for Designing Semantic Data Warehouses," in *Journal on Data Semantics XIII*. Springer, 2009, pp. 1–36.

[6] R. Cyganiak, D. Reynolds, and J. Tennison, "The RDF Data Cube Vocabulary," 2014.

[7] P. Revesz, *Introduction to Databases: From Biological to Spatio-Temporal*. Springer, 2009.

[8] J. Han, N. Stefanovic, and K. Koperski, "Selective Materialization: An Efficient Method for Spatial Data Cube Construction," in *Research and Development in Knowledge Discovery and Data Mining (PAKDD'98)*. Springer, 1998, pp. 144–158, https://dx.doi.org/10.1007/3-540-64383-4_13.

[9] E. Malinowski and E. Zimányi, *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Data-Centric Systems and Applications*. Springer, 2008, https://dx.doi.org/10.1007/978-3-540-74405-4.

[10] A. Vaisman and E. Zimányi, "A Multidimensional Model Representing Continuous Fields in Spatial Data Warehouses," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'09)*. ACM, 2009, pp. 168–177, https://doi.acm.org/10.1145/1653771.1653797.

[11] L. I. Gómez, S. A. Gómez, and A. A. Vaisman, "A Generic Data Model and Query Language for Spatiotemporal OLAP Cube Analysis," in *Proceedings of the 15th International Conference on Extending Database Technology (EDBT'12)*. ACM, 2012, pp. 300–311, https://doi.acm.org/10.1145/2247596.2247632.

[12] R. Battle and D. Kolas, "Enabling the Geospatial Semantic Web with Parliament and GeoSPARQL," *Semantic Web Journal (SWJ)*, vol. 3, no. 4, pp. 355–370, 2012, https://dx.doi.org/10.3233/SW-2012-0065.

[13] K. Kyzirakos, M. Karpathiotakis, and M. Koubarakis, "Strabon: A Semantic Geospatial DBMS," in *The Semantic Web: 11th International Semantic Web Conference (ISWC'12)*. Springer, 2012, pp. 295–311, https://dx.doi.org/10.1007/978-3-642-35176-1_19.

[14] M. Koubarakis, M. Karpathiotakis, K. Kyzirakos, C. Nikolaou, and M. Sioutis, "Data Models and Query Languages for Linked Geospatial Data," in *Reasoning Web. Semantic Technologies for Advanced Query Answering*. Springer, 2012, pp. 290–328, https://dx.doi.org/10.1007/978-3-642-33158-9_8.

[15] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "LinkedGeoData: A Core for a Web of Spatial Open Data," *Semantic Web Journal (SWJ)*, vol. 3, pp. 333–354, 2012, https://dx.doi.org/10.3233/SW-2011-0052.

[16] G. Rojas, G. Giannopoulos, and J. J. L. Daniel Hladky, "Managing Geospatial Linked Data in the GeoKnow Project," in *The Semantic Web in Earth and Space Science. Current Status and Future Directions*, vol. 20. IOS Press, 2015, p. 51.

[17] R. P. Deb Nath, K. Hose, and T. B. Pedersen, "Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses," in *Proceedings of the 18th International Workshop on Data Warehousing and OLAP (DOLAP'15)*. ACM, 2015, pp. 15–24, https://doi.org/10.1145/2811222.2811229.

[18] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Semantic Technology: 4th Joint International Semantic Technology Conference (JIST'14)*, vol. 8943. Springer, 2014, pp. 178–186, https://dx.doi.org/10.1007/978-3-319-15615-6_13.

[19] S. Bimonte, F. Johany, and S. Lardon, "A First Framework for Mutually Enhancing Chorem and Spatial OLAP Systems," in *DATA*, 2015.

[20] C. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi, "Cube algebra: A Generic User-centric Model and Query Language for OLAP Cubes," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 9, no. 2, pp. 39–65, 2013, http://dx.doi.org/10.4018/jdwm.2013040103.

Paper C

A Foundation for Spatial Data Warehouses on the Semantic Web

Nurefşan Gür, Torben Bach Pedersen, Esteban Zimányi, and Katja Hose

The paper is an extended version of Paper B. It has been published in the
Semantic Web Journal
Vol. 9, no 5, pp. 557–587, 2018. DOI: 10.3233/SW-170281

Abstract

Large volumes of geospatial data is being published on the Semantic Web (SW), yielding a need for advanced analysis of such data. However, existing SW technologies only support advanced analytical concepts such as multidimensional (MD) data warehouses and Online Analytical Processing (OLAP) over non-spatial SW data. To remedy this need, this paper presents the QB4SOLAP vocabulary which supports spatially enhanced MD data cubes over RDF data. The paper also defines a number of Spatial OLAP (SOLAP) operators over QB4SOLAP cubes and provides algorithms for generating spatially extended SPARQL queries from the SOLAP operators. The proposals are validated by applying them to a realistic use case.

1 Introduction

The Semantic Web (SW) has evolved, from focusing mostly on data publishing to also support increasingly complex queries such as interactive analytical queries. Simultaneously, the data available on the SW has evolved from being simple, most alphanumeric data, to also include complex data such as spatial data. Indeed, geospatial data is now common on the SW, but it remains difficult to analyze it.

In a non-SW context, the main tools for interactive data analyses have been Data Warehouses (DWs) and Online Analytical Processing (OLAP) tools and queries. DWs store large volumes of data and are designed with a multidimensional (MD) modeling approach, which has shown itself to be intuitive for interactive data analytics. Concretely, DWs consist of *MD data cubes*. The *cells* of the cube represent the topic of analysis, and associate observation *facts* with numerical *measures* that can be aggregated. For example, a sales fact cube has measures such as QuantitySold and SalesPrice. Facts are linked to *dimensions*, which provide contextual information, e.g., sales date, product, and location. Dimensions are perspectives, which are used to analyze data, and are organized into *hierarchies* with *levels*, e.g., Store, City, and Region, that allow users to analyze and aggregate measures at different levels of detail. Levels have a set of *attributes* that describe the characteristics of the level members.

In traditional DWs, the location dimension is widely used, but as a conventional dimension with alphanumeric data and thus only nominal reference to spatial concepts such as areas and places. This does not allow manipulating through spatial location data or deriving topological relations among the hierarchy levels of the location dimension. This yields a demand for truly spatial DWs for better analysis purposes. Including the geometric information of the location data, significantly improves the analysis process (i.e., proximity analysis of the locations) with additional perspectives by revealing dynamic spatial hierarchy levels and new spatial members.

Similarly, providing deep spatial analytics support for spatial SW data is very valuable. Spatial data requires specific treatment techniques, in particular encoding, special functions and different manipulation methods, which should be considered in the modeling process and querying. The current state of the art for the geospatial Semantic Web focuses on techniques for publishing, linking and querying spatial data, but supports only "plain" spatial SW data (without support for spatial DW concepts such as spatial hierarchies, levels, and measures) and does not consider analytical queries over spatial RDF data (see Section 2 for details).

Problem Definition. The proliferation of open geospatial data on the SW creates possibilities for advanced analysis of such data. Many examples exist

of spatial Linked Open Data (LOD) published on the SW as RDF[1,2,3,4]. These datasets have observations and measures that are well suited for analytical queries (e.g., water/air quality measurements, immigration rates, EU subsidies in agriculture, crop revenue, etc.). However, such datasets are typically not modeled with spatial dimension levels and hierarchies. Thus, they cannot be queried with interactive spatial analytical queries (a.k.a. SOLAP) on the SW. In the current state of the SW, if a (spatial) DW user would like to query the existing spatial RDF data from the SW with SOLAP operations, the user needs to download the RDF data, map it to a relational data model (i.e., with a snowflake schema), and then import it into a traditional spatial data warehouse in order to query with SOLAP, which is slow, labor-intensive, and stores the data in a non-open format.

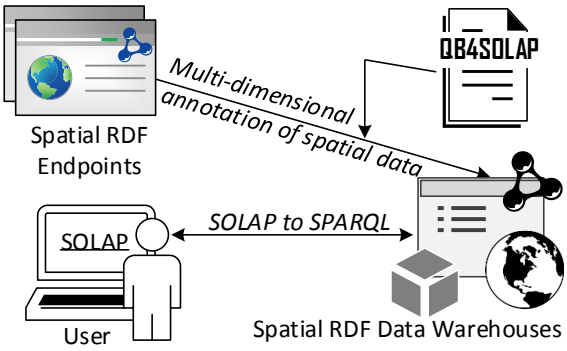


Fig. C.1: QB4SOLAP approach to SOLAP on the SW

Our Approach. On the contrary, annotating spatial RDF datasets with QB4SOLAP allows users to define spatial multidimensional concepts on top of existing RDF data [1, 2]. Hence, the user can create and publish spatial data warehouses on the Semantic Web, which can be easily queried with SOLAP operations. Fig. C.1 depicts the general workflow scenario, where the spatial RDF datasets from endpoints can be annotated with QB4SOLAP. This makes it possible for end users to use SOLAP queries. However, writing a SOLAP query in SPARQL can be very complicated for users inexperienced with SPARQL (e.g., traditional DW users). Due to the lack of MD semantics of spatial RDF data and the lack of translation techniques from high-level SOLAP expressions to SPARQL, there is a considerable entry barrier for advanced spatial data analysis on the SW for data warehouse users.

[1]EuroStat: `http://ec.europa.eu/eurostat`
[2]UK Environmental Data: `http://environment.data.gov.uk`
[3]Danish Agricultural Data: `https://datahub.io/dataset/govagribus-denmark`
[4]Australian Climate Observations: `https://datahub.io/dataset/acorn-sat`

Contributions. In order to address these issues, this paper makes a number of contributions. First, we propose QB4SOLAP, a generic and extensible vocabulary (metamodel) for spatial DWs on the SW. QB4SOLAP extends the most recent stable version of the QB4OLAP vocabulary with spatial concepts. We provide a full formalization of QB4SOLAP. The key concepts of spatial cube members, spatial hierarchies and levels, spatial measures, spatial aggregate functions (e.g., union, buffer, and convex–hull) and topological relations among spatial dimension and hierarchy level members (e.g., within, intersects, and overlaps), are defined. Second, we define a number of analytical Spatial OLAP (SOLAP) operators over the model including giving formal semantics of the operators. The operators support advanced analytical queries over MD geospatial SW data. Third, we provide algorithms for generating spatially extended SPARQL queries for individual and nested SOLAP operators, which allows writing SOLAP queries without knowledge of RDF/SPARQL. Fourth, we validate the vocabulary, operators, and query generation algorithms by applying them to a realistic use case.

Paper structure. The remainder of the paper is structured as follows. Section 2 discusses related work. Section 3 defines preliminary spatial and OLAP concepts. Section 4 defines the QB4SOLAP vocabulary, while Section 5 defines the SOLAP operators. Section 6 provides the SPARQL query generation algorithms. Finally, Section 7 concludes the paper and points to future research.

2 Related work

DW and OLAP technologies have been successful for analyzing large volumes of data [3]. Combining DW/OLAP technologies with RDF data makes RDF data sources more easily available for interactive analysis. The following work concerns the integration of DW/OLAP with the SW.

DW/OLAP and Semantic Web. Using OLAP to analyze SW data is considered in several approaches. Kämpgen et al. propose an extended model [4] on top of the RDF Data Cube Vocabulary (QB) [5] for interacting with statistical linked data via OLAP operations directly in SPARQL. However, it has the inherent limitations of QB and thus cannot support OLAP dimensions with hierarchies and levels, and built-in aggregate functions. Etcheverry et al. introduce QB4OLAP [6] as an extended vocabulary based on QB, with a full MD metamodel, supporting OLAP operations directly over RDF data with SPARQL queries. Nath et al. considers creating an Extract–Transform–Load (ETL) framework for semantic data warehouses [7]. Varga et al. presents a comprehensive methodology for dimensional enrichment of statistical LOD by using QB4OLAP and provide a SW-based OLAP engine for traditional

DW users [8]. However, these approaches and vocabularies support neither spatial DWs nor provide SOLAP operators for the SW.

Spatial DW and OLAP. The constraint representation of spatial data has been the focus in many fields from databases to AI [9]. Extending OLAP with spatial features has also attracted the attention of the data warehousing community. Bédard et al. first introduced the term SOLAP [10] in 1997. SOLAP systems [11, 12] since then, have significantly been improved. Respectively, various papers improve the spatial aggregation functions and techniques [13–16].

Several conceptual models are proposed for representing spatial data in data warehouses. Stefanovic et al. [17] considers constructing and materializing spatial cubes in their proposed model. The MultiDim conceptual model, introduced by Malinowski and Zimányi [18], copes with spatial features and is extended in [19], to include complex geometric features (continuous fields), with a set of operations and an MD calculus supporting spatial data types. Gómez et al. [20] propose an algebra and a general framework for OLAP cube analysis on discrete and continuous spatial data. Even though spatial data warehousing is thus widely studied, those studies are limited to traditional *non-semantic* spatial data warehouses and SOLAP techniques. The work above neither considered semantic web data nor spatial analytical querying in SPARQL.

Geospatial Semantic Web. The Open Geospatial Consortium (OGC) has proposed GeoSPARQL [21] as a vocabulary to represent and query spatial data in RDF using an extension to SPARQL. Kyzirakos et al. present a comprehensive survey of data models and query languages for linked geospatial data in [22], and propose a semantic geospatial data store called Strabon in [23]. Strabon has an extensive query language called stSPARQL , which is however limited to the specific environment. LinkedGeoData is a significant contribution on interactive transformation of OpenStreetMap[5] data to RDF data [24]. GeoKnow [25] is a more recent project with focus on linking geospatial data from heterogeneous sources. Andersen et al. considers publishing/converting open spatial data as Linked Open Data [26]. However, none of these works consider the MD aspects of geospatial data or allow querying with SOLAP on the SW, unlike QB4SOLAP. The QB4SOLAP vocabulary is validated with both the running example use case, the GeoNorthwind data cube, as well as a substantial real-world use case, the *GeoFarmHerdState* data cube [2]. GeoFarmHerdState is a spatial data cube about livestock holdings in Denmark, which integrates environmental and geographical open data from several sources, thus enabling a range of interesting SOLAP queries.

In summary, none of the related work, which is surveyed in the fields of “DW/OLAP and the SW", “Spatial DW and OLAP", and “Geospatial Semantic Web" provides a substantial foundation for modeling and querying

[5] http://www.openstreetmap.org

spatial data warehouses on the Semantic Web, unlike the QB4SOLAP vocabulary, SOLAP operators, and SPARQL generation algorithms presented in this paper.

3 Preliminary concepts

In this section, we describe the spatial objects and the spatial operations that manipulate them. Then, we introduce the data cubes and spatial enhancement on them as spatial data cubes. Finally, we show the traditional OLAP operations, which manipulate data cubes, and explain the Spatial OLAP (SOLAP) operators, which manipulate *spatial* data cubes.

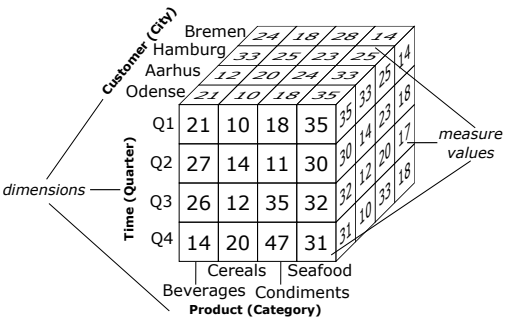


Fig. C.2: A three-dimensional cube for Sales data

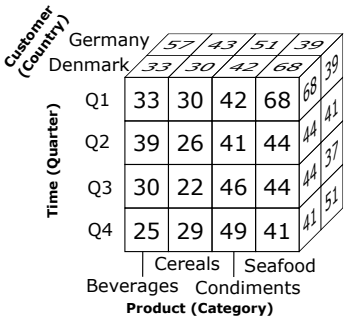


Fig. C.3: Roll-up to the Country level

3.1 Spatial objects

A spatial object represents a real-world object whose geographic features are important for an application. These geographic features are encoded using the *geometry* data type. *Point, Line, and Polygon* are the basic instantiable types of the geometry data type. Coordinates for geometry data type are generally given in 2-dimensions with X, Y values. Geometries are associated with a *spatial reference system* (SRS), which describes the coordinate space in which the geometry is defined. There are several SRSs and each of them are identified with a *spatial reference system identifier* (SRID). The World Geodetic System (WGS) is the most well-known SRS and the latest version is called WGS84, which is also used in our use case.

3.2 Spatial operations

There is a set of spatial operations that can be applied on spatial data. We grouped these operations into classes, based on the common functionality of

the operators. These classes are defined next.

Definition 1. (Spatial aggregation) The operators in the spatial aggregation class $\mathcal{S}_{agg}$ aggregate two or more spatial objects and return a new spatial object. Union, Intersection, ConvexHull, and MinimumBoundingRectangle (MBR) are example operators of this class. Some spatial functions such as ConvexHull or MBR can also be interpreted as unary spatial functions with a single parameter, but here we only consider the aggregate versions of the functions. In order to make this clear, the aggregate versions of those functions are given with a prefix "*Aggr*" in the QB4SOLAP vocabulary (Fig. C.6). For our purpose, it is enough to group all spatial aggregate functions into a single group, although more fine-grained classification proposals for spatial aggregate functions exist [15].

Definition 2. (Topological relations) The operators in the topological relation class $\mathcal{T}_{rel}$ are commonly expressed in the RCC8[6] and DE-9DIM[7] models [27, 28]. Topological relations are Boolean predicates that specify how two spatial objects are related to each other. Examples of topological relations are Intersects, Disjoint, Equals, Overlaps, Contains, Within, Touches, Covers, CoveredBy, and Crosses.

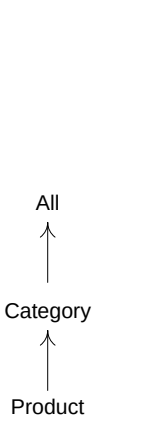
Definition 3. (Numeric operations) The operators in the numeric operation class $\mathcal{N}_{op}$ take one or more spatial objects and return a numeric value. Perimeter, Area, NoOfInteriorRings, Distance, HaversineDistance, and NoOfGeometries are example operators of this class.

3.3 Data cubes

Data warehouses store large volumes of data for decision support. They are based on the multidimensional model, which views data in an n-dimensional space, usually called a *data cube*. The cells of the cube represent the observation *facts* for analysis with a set of attributes called *measures* (e.g., a sales fact cube with measures product quantity and price). Facts are linked to *dimensions*, which provide perspectives to analyze data (e.g., sales date, product, and customer location). Dimensions are organized into *hierarchies*, which allow users to aggregate measures at various levels of detail. Hierarchies are composed of *levels* and there is always a unique top level *All* with just one member *all*. Levels have a set of *attributes* that describe the characteristics of the level members.

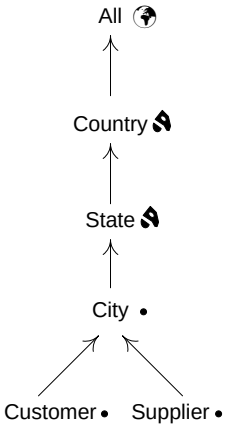
[6]RCC8 (Region Connection Calculus) describes regions in Euclidean space or in a topological space by their possible relations to each other.

[7]DE-9DIM (Dimensionally Extended Nine-Intersection Model) is a topological model that describes spatial relations of two geometries in two dimensions.

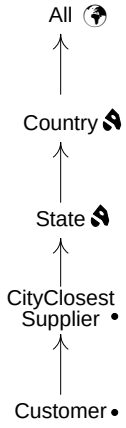


(a) Categories hierarchy in the Product dimension

(b) Calendar hierarchy in the Time dimension



(c) Spatial Geography hierarchies in the Customer and Supplier dimensions



(d) Dynamic spatial Geography hierarchy in the Customer dimension

Fig. C.4: Dimension hierarchies

An example of a data cube with three dimensions (Customer, Time, and Product) and one measure (Quantity) is given in Fig. C.2. Each cell in the cube is an observation fact, which is characterized by dimension and measure values. The hierarchies of this cube are given in Fig. C.4(a)–(c). Thus, in the cube shown in Fig. C.2, the Product dimension is given at the Category level, the Time dimension at the Quarter level, and the Customer dimension at the City level. Measure values represent the measure Quantity of the sold products.

3.4 Spatial data cubes

A spatial data cube contains both conventional and spatial dimensions. A *spatial dimension* is a dimension, which includes at least one *spatial level* in which the application should store the spatial characteristics of the members. Similarly, a hierarchy is a *spatial hierarchy* if it has at least one *spatial level*. Spatial characteristics of the levels are captured by their geometries and can be recorded in the *spatial attributes* of the level. A *spatial fact* is a fact that relates several dimensions in which, two or more are spatial.

For example, consider a Sales *spatial fact*, which has *spatial dimensions* Customer and Supplier, each with a *spatial hierarchy* Geography composed of *spatial levels* City $\rightarrow$ State $\rightarrow$ Country $\rightarrow$ *All* (Fig. C.4(c)). These *spatial levels* record the spatial characteristics of its members with *spatial attributes*: Customer, Supplier, and City using a *point* spatial data type, whereas State and Country with a *multi-polygon* spatial data type.

Following the rules of spatially extended MultiDim conceptual model [19], MD concepts such as levels are considered to be spatial, only if they record the spatial characteristics of the concepts as geometries. For instance, "continent" might be considered as a spatial object, in theory or in other vocabularies. However, if there is no information about the geometry of the continents in the schema and in the instance data, continent does not become a spatial level (Ext. 7), although continent might still be a traditional level (Def. 7) of the spatial hierarchy Geography with alphanumeric attributes (i.e., continent name, code, and etc.).

Spatial data cubes typically have *spatial measures*, which are also represented by a spatial data type. An example is a SalesPoint measure that stores the location of sales. Fig. C.7 shows the multidimensional schema of the GeoNorthwind data warehouse, which is used as running example in the paper.

3.5 OLAP operators

OLAP operators are used for expressing queries over data cubes. The traditional OLAP operators are given next.

The *slice* operator removes a dimension from a cube by selecting one instance in a dimension level. An example is "slice on City is equal to Odense".

The *dice* operator selects the cells in a cube that satisfy a Boolean condition. An example is "dice on the first and last quarter of the year".

The *roll-up* operator aggregates measures along a hierarchy to obtain data at a coarser granularity. An example is "roll-up to the Country level" (Fig. C.3).

Finally, the *drill-down* operator disaggregates measures along a hierarchy to obtain data at a finer granularity. It is the inverse operation of roll-up. Starting from the cube in Fig. C.3, an example is "drill-down to the City

level".

3.6 Spatial OLAP operators

Spatial OLAP (SOLAP) operates on *spatial* data cubes. SOLAP increases the analytical capabilities of OLAP by taking into account the spatial information in the cube. SOLAP operators involve *spatial conditions* or *spatial functions* by using the spatial operators defined in Sect. 3.1. Spatial conditions specify constraints on the geometries associated to cube members or measures, while spatial functions derive new data from the cube, which can be used, e.g., to derive dynamic spatial hierarchies or levels, as explained in the following example. Spatial extensions of the common OLAP operators are formally defined in Sect. 5.

Table C.1: Sample (instance) data for the Sales cube

Customer City		**Supplier**			**Total**
Customer		s1	s2	s3	**Sales**
Düsseldorf	c1	8	–	3	11
	c2	10	–	–	10
Dortmund	c3	7	4	–	11
	c4	–	20	3	23
Münster	c5	–	–	30	30

Table C.2: Roll-up of the Sales cube

Customer City	**Sales**
Düsseldorf	21
Dortmund	34
Münster	30

Table C.3: S-Roll-up of the Sales cube

CityClosest Supplier	**Sales**
Düsseldorf	25
Dortmund	20
Münster	33

Table C.4: Customer to Supplier distance

		Supplier City		
		Supplier		
Customer City		Düsseldorf	Dortmund	Münster
Customer		s1	s2	s3
Düsseldorf	c1	15 km	45 km	30 km
	c2	15 km	60 km	60 km
Dortmund	c3	15 km	30 km	45 km
	c4	45 km	15 km	15 km
Münster	c5	60 km	45 km	15 km

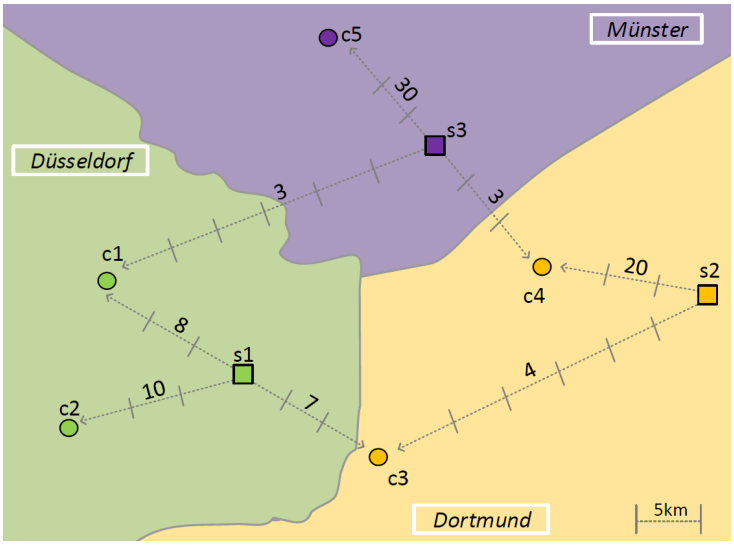


Fig. C.5: Example map of Sales (instance) data

Example 1. Consider the summarized data for the Sales cube given in Table C.1, where a '–' is used if there are no sales to customers from the corresponding suppliers. The data in Table C.1 is shown on the map in Fig. C.5, where the arrows on the map between the supplier and customer locations represent the distance. The quantities of sold products are shown along these arrows.

The hierarchies in Fig. C.4(a)–(c) can be used to perform classical roll-up operations, where measures are aggregated from a child to a parent level. An example of such a roll-up operator is expressed by the query "total sales to customers by city", whose results is given in Table C.2.

On the other hand, as shown in Table C.4 and Fig. C.5, some customers may be closer to suppliers from other cities. For example, customer c3 is related to its city Dortmund by using traditional Geography hierarchy, but the customer is closer to the city Düsseldorf of supplier s1. Similarly, customer c4 in city Dortmund is closer to the city Münster of supplier s3. Fig. C.4(d) shows a new *dynamic spatial hierarchy* that can be obtained with a spatial roll-up (s-roll-up) operator that expresses the query "total sales to customers by city of the closest supplier". Such queries are not possible to express on conventional hierarchies with traditional OLAP.

The hierarchy in Fig. C.4(d) is created on the fly with the help of a spatial function computing the distance between customer and supplier locations. Therefore, using the s-roll-up operator, sales to customers are aggregated by city of the closest suppliers, where Dortmund has a significant drop off in

the quantity of the sales from 34 (Table C.2) to 20 (Table C.3).

4 The QB4SOLAP vocabulary

In this section, we formally define how to represent (spatial) data cubes in RDF. We use as running example the GeoNorthwind data warehouse whose conceptual schema is given in Fig. C.7.

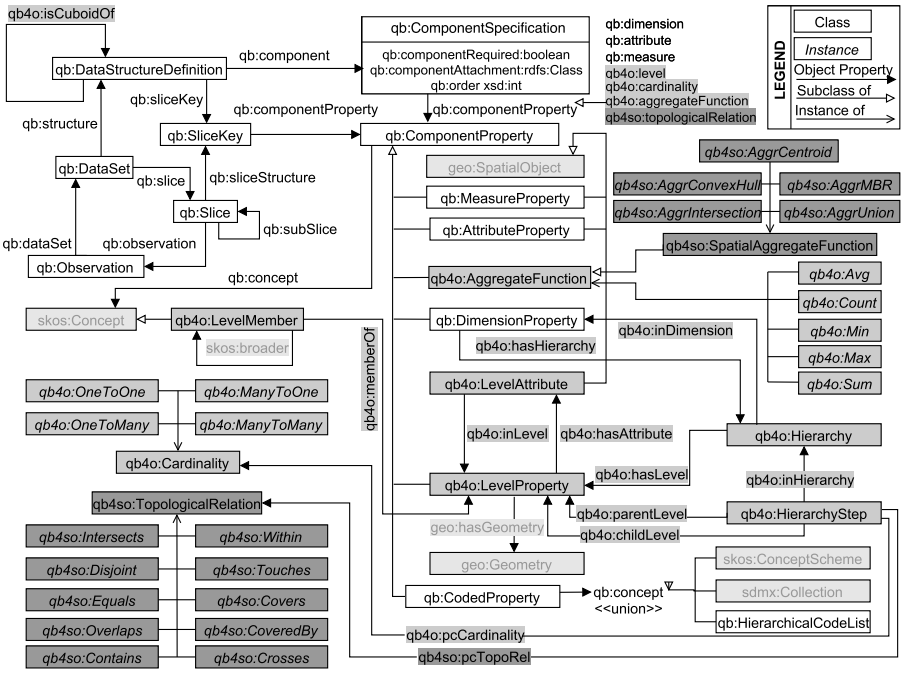


Fig. C.6: The QB4SOLAP vocabulary

The QB4OLAP [6] vocabulary allows to define *cube schemas* and *cube instances* as RDF triples. QB4OLAP is an extension of the RDF Data Cube Vocabulary (QB) [5] with multidimensional concepts in order to be able to support OLAP operations directly over RDF data with SPARQL queries. We extended QB4OLAP (v1.2)[8] with spatial concepts to give QB4SOLAP [1]. We based our extension on GeoSPARQL [29], a standard from the Open Geospatial Consortium (OGC) for representing and querying geospatial linked data for the Semantic Web. Since our base vocabulary QB4OLAP uses the MultiDim conceptual model to describe the multidimensional concepts, we base our definitions on a spatially extended version of MultiDim model [19] for

[8]QB4OLAP v1.2: `https://github.com/lorenae/qb4olap/blob/master/rdf/qb4olap.1.2.ttl`

spatial extension of the MD concepts. Fig. C.6 shows the QB4SOLAP vocabulary for representing a spatial cube schema and spatial cube members as RDF triples. A cube schema defines the structure of the cube in terms of dimension levels, measures, aggregation functions (e.g., SUM, AVG, COUNT) on measures, spatial aggregation functions ($\mathcal{S}_{agg}$ in Def. 1) on spatial measures, dimensions hierarchies, and parent–child relationships between levels (including their cardinality and topological relationships for spatial levels). These schema level metadata are used to define multidimensional datasets in RDF. Cube members are the instances of a cube schema that represent level members, facts, and measure values. As we will show in Sect. 6, we use the schema level metadata to produce SPARQL queries that implement SOLAP operators on cube members.

Terms with capitalized initials and non-italic font in Fig. C.6 represent RDF classes, terms with capitalized initials and italic font represent RDF instances, and terms with non-capitalized initials represent RDF properties. Classes in external vocabularies are depicted in light gray background and font. RDF Cube (QB), QB4OLAP, and QB4SOLAP classes are shown, respectively, with white, light gray, dark gray backgrounds. Original QB terms are prefixed with `qb:`[9]. QB4OLAP and QB4SOLAP terms are prefixed, respectively, with `qb4o:`[10] and `qb4so:`[11]. Spatial classes and properties are prefixed with `geo:`[12].

In what follows, we first define formally RDF triples, and then discuss how to describe (spatial) multidimensional data using the QB4OLAP Vocabulary and QB4SOLAP Vocabulary.

Definition 4. (RDF triple) An *RDF triple* $t = (s, p, o)$ consists of three components: s is the subject, p is the predicate, and o is the object. RDF triples are defined over

$$\mathcal{T} = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$$

where $\mathcal{I}$ is the set of *IRIs*, $\mathcal{B}$ is the set of *blank nodes*, and $\mathcal{L}$ is the set of *literals*.

A set of RDF triples is referred to as a graph. We denote a QB4SOLAP graph by $\mathcal{G}$, where $\mathcal{G} \subset \mathcal{T}$.The cube schema and cube instances are subsets of this graph and are denoted, respectively, by $\mathcal{G}^S$ and $\mathcal{G}^I$, where $\mathcal{G}^S \subset \mathcal{G}$ and $\mathcal{G}^I \subset \mathcal{G}$.

Given an MD element $x \in (\mathcal{I} \cup \mathcal{B})$ in a schema graph or instance graph $\mathcal{G}$, we define by $\mathcal{G}_x$ the subgraph of $\mathcal{G}$ for x, where $\mathcal{G}_x \subset \mathcal{G}$. We define the function $id(x) : \mathcal{G} \rightarrow \mathcal{I}$, that given a MD element x returns its identifier $\mathcal{I}$ from the graph $\mathcal{G}$. We use superscript notation to indicate the type of the identifier

[9]RDF cube: `http://purl.org/linked-data/cube#`
[10]QB4OLAP: `http://purl.org/qb4olap/cubes#`
[11]QB4SOLAP: `http://w3id.org/qb4solap#`
[12]GeoSPARQL:`http://www.opengis.net/ont/geosparql#`

from the cube schema graph ($\mathcal{G}^S$) and cube instance graph ($\mathcal{G}^I$), e.g., $id^S(x)$ for a cube schema identifier and $id^I(x)$ for a cube instance identifier.

4.1 Defining spatial data cube schemas with QB4SOLAP

An *n-dimensional cube schema* $\mathcal{CS}$ is a tuple $\mathcal{CS} = (D, M, F)$, with a set of dimensions D, a set of measures M, and a fact F. A dimension $d \in D$ has a set of hierarchies $H(d)$. Each hierarchy $h \in H(d)$ is organized into a set of levels $L(h)$. Each level $l \in L(h)$ has a set of attributes $A(l)$. Each attribute $a \in A(l)$ is defined over a domain. Each measure $m \in M$ is also defined over a domain.

We define next how to represent a cube schema $\mathcal{CS}$ in RDF by using the QB4SOLAP Vocabulary. We denote the RDF graph of the cube schema $\mathcal{G}^S$. In the examples we prefix the elements of $\mathcal{G}^S$ with `gnw:`. We follow a similar naming convention for schema elements as in QB4OLAP. If there is a possibility of confusion for different MD concepts with same schema name, i.e., customer dimension and customer level, we suffix the dimensions with `Dim` (e.g., `gnw:customerDim` for dimension, and `gnw:customer` for level). The subgraph of $\mathcal{G}^S$ that refers to a specific schema element x is denoted by $\mathcal{G}^S_x$ and the unique identifier of x is denoted by $id^S(x)$.

Definition 5. (Dimensions) An n-dimensional cube schema $\mathcal{CS}$ has a set of dimensions $D = \{d_1, \ldots, d_n\}$ and each dimension d_i has a set of hierarchies $H(d_i)$ (Def. 6). Each dimension $d_i \in D$ is defined in the cube schema graph $\mathcal{G}^S$ with `qb:DimensionProperty`. Each hierarchy $h \in H(d_i)$ is linked to its dimension d_i with the `qb4o:hasHierarchy` property. The RDF graph formulation of the dimensions D is represented as

$$\mathcal{G}^S_D = \bigcup_{i=1}^{n} \mathcal{G}^S_{d_i}$$

where

$$\mathcal{G}^S_{d_i} = \{(id^S(d_i)\ \texttt{rdf:type}\ \texttt{qb:DimensionProperty})\} \cup \bigcup_{h \in H(d_i)} \{(id^S(d_i)\ \texttt{qb4o:hasHierarchy}\ id^S(h))\}$$

Extension 5. (Spatial dimensions) A dimension is spatial if it has at least one spatial level. A spatial dimension d_{i_s} belongs to the set of spatial dimensions D_s, which is a subset of the set of dimensions D, such that $d_{i_s} \in D_s \subseteq D$.

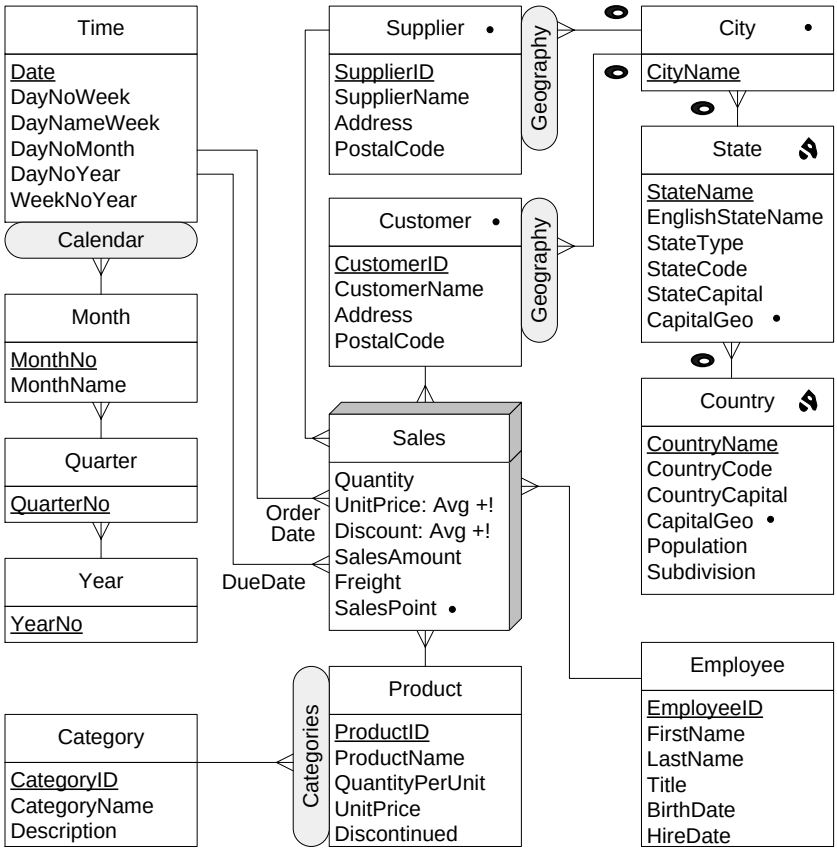


Fig. C.7: Conceptual multidimensional schema of the GeoNorthwind data warehouse

Example 2. The triples below show how some of the dimensions of the GeoNorthwind DW (Fig. C.7) are represented in RDF using Def. 5 and Ext. 5. As we will see below, the Customer and Supplier dimensions are spatial as they both have a spatial hierarchy Geography.

```
# Dimensions
gnw:customerDim rdf:type qb:DimensionProperty ;
   qb4o:hasHierarchy gnw:customerGeography .
gnw:supplierDim rdf:type qb:DimensionProperty ;
   qb4o:hasHierarchy gnw:supplierGeography .
gnw:productDim rdf:type qb:DimensionProperty ;
   qb4o:hasHierarchy gnw:categories .
gnw:employeeDim rdf:type qb:DimensionProperty .
```

Definition 6. (Hierarchies) A dimension d has a set of hierarchies $H(d) = \{h_1, \ldots, h_m\}$, where each hierarchy h_i has a set of levels $L(h_i)$ (Def. 7). Each hi-

erarchy $h_i \in H(d)$ is defined in the cube schema graph $\mathcal{G}^S$ with the `qb4o:Hierarchy` predicate and is linked with its dimension d by the `qb4o:inDimension` property. Each level $l \in L(h_i)$ that belongs to a hierarchy h_i is defined with the `qb4o:hasLevel` property. The RDF graph formulation of the hierarchies $H(d)$ is represented as

$$\mathcal{G}^S_{H(d)} = \bigcup_{i=1}^{m} \mathcal{G}^S_{h_i}$$

where

$$\begin{aligned}\mathcal{G}^S_{h_i} = \{&(id^S(h_i)\ \texttt{rdf:type}\ \texttt{qb4o:Hierarchy})\} \cup \\ &\{(id^S(h_i)\texttt{qb4o:inDimension}\ id^S(d))\}\cup \\ &\bigcup_{l \in L(h_i)} \{(id^S(h_i)\ \ \texttt{qb4o:hasLevel}\ id^S(l))\}\end{aligned}$$

Extension 6. (Spatial hierarchies) A hierarchy is spatial if it contains at least one spatial level. A spatial hierarchy h_{i_s} belongs to the set of spatial hierarchies $H_s(d)$, which is a subset of the set of hierarchies $H(d)$, such that $h_{i_s} \in H_s(d) \subseteq H(d)$.

Example 3. The triples below show how some of the hierarchies from the GeoNorthwind DW (Fig. C.7) are represented in RDF using Def. 6 and Ext. 6. As we will see below, the Geography hierarchies in the Customer and Supplier dimensions are spatial since they have spatial levels (City, State, etc.)

```
# Hierarchies
gnw:customerGeography rdf:type qb4o:Hierarchy ;
   qb4o:inDimension gnw:customerDim ;
   qb4o:hasLevel gnw:customer, gnw:city,
      gnw:state, gnw:country .
gnw:supplierGeography rdf:type qb4o:Hierarchy ;
   qb4o:inDimension gnw:supplierDim ;
   qb4o:hasLevel gnw:supplier, gnw:city,
      gnw:state, gnw:country .
gnw:categories rdf:type qb4o:Hierarchy ;
   qb4o:inDimension gnw:productDim ;
   qb4o:hasLevel gnw:product, gnw:category .
```

Definition 7. (Levels) A hierarchy h has a set of levels $L(h) = \{l_1, \ldots, l_k\}$ and each level l_i has a set of attributes $A(l_i)$ (Def. 8). Each level $l_i \in L(h)$ is defined in the cube schema graph $\mathcal{G}^S$ with the `qb4o:LevelProperty` predicate. Each attribute $a \in A(l_i)$ is linked to its level l_i with the `qb4o:hasAttribute` property. The RDF graph formulation of the levels $L(h)$ is represented as

$$\mathcal{G}^S_{L(h)} = \bigcup_{i=1}^{k} \mathcal{G}^S_{l_i}$$

where

$$\mathcal{G}_{l_i}^{S} = \{(id^{S}(l_i)\ \texttt{rdf:type qb4o:LevelProperty})\} \cup \bigcup_{a \in A(l_i)} \{(id^{S}(l_i)\ \texttt{qb4o:hasAttribute}\ id^{S}(a))\}$$

Extension 7. (Spatial levels) A level is spatial if it has an associated geometry. A spatial level l_{i_s} belongs to the set of spatial levels $L_s(h)$, which is a subset of the set of levels $L(h)$, such that $l_{i_s} \in L_s(h) \subseteq L(h)$. The geometry of a spatial level is defined in the cube schema graph $\mathcal{G}^S$ with the `geo:hasGeometry` property The RDF graph formulation of the spatial levels $L_s(h)$ is represented as

$$\mathcal{G}_{L_s(h)}^{S} = \bigcup_{i=1}^{k} \mathcal{G}_{l_{i_s}}^{S}$$

where

$$\mathcal{G}_{l_{i_s}}^{S} = \{(id^{S}(l_{i_s})\ \texttt{rdf:type qb4o:LevelProperty})\} \cup \{(id^{S}(l_{i_s})\ \texttt{geo:hasGeometry geo:Geometry})\} \cup \bigcup_{a \in A(l_{i_s})} \{(id^{S}(l_{i_s})\ \texttt{qb4o:hasAttribute}\ id^{S}(a))\}$$

Example 4. The triples below show how some of the levels of the GeoNorthwind DW (Fig. C.7) are represented in RDF using Def. 7 and Ext. 7. Note that the Customer and City levels are spatial as they have a geometry that is specified at the level definition.

```
# Levels
gnw:customer rdf:type qb4o:LevelProperty ;
   qb4o:hasAttribute gnw:customerID ;
   qb4o:hasAttribute gnw:customerName ;
   qb4o:hasAttribute gnw:address ;
   qb4o:hasAttribute gnw:postalCode ;
   geo:hasGeometry gnw:customerGeometry .
gnw:city rdf:type qb4o:LevelProperty ;
   qb4o:hasAttribute gnw:cityName ;
   geo:hasGeometry gnw:cityGeometry .
```

Definition 8. (Attributes) A level l has a set of attributes $A(l) = \{a_1, \ldots, a_p\}$, which defines the characteristics of the level members. One among these attribute, denoted as a_{ID}, specifies a surrogate key for the level, i.e., the value of a_{ID} uniquely identifies the members of the level. For simplicity, we assume that it is the first attribute in the set of attributes $A(l)$, i.e.,

$a_1 = a_{ID}$. Each attribute $a_i \in A(l)$ is defined in the cube schema graph $\mathcal{G}^S$ with the `qb4o:LevelAttribute` predicate and is linked to its level l with the `qb4o:inLevel` property. Each attribute a_i is defined as ranging over XSD literals $\mathcal{L}$ using the `rdfs:range` property. The RDF graph formulation of the attributes $A(l)$ is represented as

$$\mathcal{G}^S_{A(l)} = \bigcup_{i=1}^{p} \mathcal{G}^S_{a_i}$$

where

$$\begin{aligned}
&\mathcal{G}^S_{a_i} = \\
&\quad \{(id^S(a_i)\ \texttt{rdf:type}\ \texttt{qb4o:LevelAttribute})\} \cup \\
&\quad \{(id^S(a_i)\ \ \texttt{qb4o:inLevel}\ \ id^S(l))\} \cup \\
&\quad \{(id^S(a_i)\ \ \texttt{rdfs:range}\ \ \mathcal{L})\}
\end{aligned}$$

Extension 8. (Spatial attributes) An attribute is spatial if it is defined over a spatial domain. A spatial attribute a_{i_s} belongs to the set of spatial attributes $A_s(l)$, which is a subset of the set of attributes $A(l)$, such that $a_{i_s} \in A_s(l) \subseteq A(l)$. The RDF graph formulation of the spatial attributes is similar as in Def. 8. However, the attribute must range over spatial literals $\mathcal{L}_s$ i.e., a well-known text literal (WKT) from OGC schemas. Further, the domain of the attribute should be specified with the `rdfs:domain` property, which must be a geometry. Finally, the attribute must be specified as spatial object with the `rdfs:subclassOf` property. The RDF graph formulation of the spatial attributes $A_s(l)$ is represented as

$$\mathcal{G}^S_{A_s(l)} = \bigcup_{i=1}^{p} \mathcal{G}^S_{a_{i_s}}$$

where

$$\begin{aligned}
&\mathcal{G}^S_{a_{i_s}} = \\
&\quad \{(id^S(a_{i_s})\ \texttt{rdf:type}\ \texttt{qb4o:LevelAttribute})\} \cup \\
&\quad \{(id^S(a_{i_s})\ \ \texttt{qb4o:inLevel}\ \ id^S(l))\} \cup \\
&\quad \{(id^S(a_{i_s})\ \ \texttt{rdfs:range}\ \ \mathcal{L}_s)\} \cup \\
&\quad \{(id^S(a_{i_s})\ \texttt{rdfs:subPropertyOf}\ \texttt{geo:Geometry})\} \cup \\
&\quad \{(id^S(a_{i_s})\ \texttt{rdfs:subClassOf}\ \texttt{geo:SpatialObject})\}
\end{aligned}$$

Example 5. The triples below show how some of the attributes of the GeoNorthwind DW (Fig. C.7) are represented in RDF using Def. 8 and Ext. 8. Note that

the Customer level has a spatial attribute (Customer geometry). It is represented as a WKT literal that defines a Point type from the Geometry class, which is a subclass of Spatial Object.

```
# Attributes
gnw:customerID rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:Integer .
gnw:customerName rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:String .
gnw:address rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:String .
gnw:postalCode rdf:type qb4o:LevelAttribute ;
   qb4o:inLevel gnw:customer;
   rdfs:range xsd:String .
gnw:customerGeometry rdf:type
qb4o:LevelAttribute ;
   rdfs:subPropertyOf geo:Geometry ;
   qb4o:inLevel gnw:customer ;
   rdfs:range geo:wktLiteral;
   rdfs:domain geo:Point ;
   rdfs:subClassOf geo:SpatialObject .
```

Definition 9. (Hierarchy steps) A hierarchy h has a set of hierarchy steps $HS(h) = \{hs_1, \ldots, hs_q\}$, which define the structure of the hierarchy in relation with its corresponding levels. A hierarchy step $hs_i = (l_c, l_p, card) \in HS(h)$ entails a roll-up relation between a lower (child) level l_c to an upper (parent) level l_p with a cardinality *card*. The cardinality $card \in \{1\text{-}1, 1\text{-}n, n\text{-}1, n\text{-}n\}$ describes the number of members in one level that can be related to a member in the other level for both the child and the parent levels.

Each hierarchy step hs_i is defined in the cube schema graph $\mathcal{G}^S$ as a blank node `_:hs`$_i \in \mathcal{B}$ with the `qb4o:HierarchyStep` predicate. Each hierarchy step is linked to its hierarchy with the `qb4o:inHierarchy` property. The child and parent levels are linked in a hierarchy step with the `qb4o:childLevel` and `qb4o:parentLevel` properties, respectively. The cardinality *card* of a hierarchy step is defined by the `qb4o:pcCardinality` property. The RDF graph formulation of the hierarchy steps $HS(h)$ is represented as

$$\mathcal{G}^S_{HS(h)} = \bigcup_{i=1}^{q} \mathcal{G}^S_{hs_i}$$

where

$$\mathcal{G}^S_{hs_i} = \{(\_{:}\mathtt{hs}_i\ \mathtt{rdf{:}type}\ \mathtt{qb4o{:}HierarchyStep})\} \cup \{(\_{:}\mathtt{hs}_i\ \mathtt{qb4o{:}inHierarchy}\ id^S(h))\} \cup \{(\_{:}\mathtt{hs}_i\ \mathtt{qb4o{:}parentLevel}\ id^S(l_p))\} \cup \{(\_{:}\mathtt{hs}_i\ \mathtt{qb4o{:}childLevel}\ id^S(l_c))\} \cup \{(\_{:}\mathtt{hs}_i\ \mathtt{qb4o{:}pcCardinality}\ id^S(card))\}$$

Extension 9. (Spatial hierarchy steps) A hierarchy step is spatial if it relates a spatial child level l_{c_s} and a spatial parent level l_{p_s}, in which case it entails a topological relationships between these spatial levels. A spatial hierarchy step is then a tuple $hs_{i_s} = (l_{c_s}, l_{p_s}, card, topoRel)$ where the topological relation $topoRel$ belongs to the $\mathcal{T}_{rel}$ class (Def. 2). The topological relation between parent-child levels of a spatial hierarchy step is defined by the `qb4so:pcTopoRel` property. The RDF graph formulation of the spatial hierarchy steps $HS_s(h)$ (w.r.t. Def. 9) is represented as

$$\mathcal{G}^S_{HS_s(h)} = \bigcup_{i=1}^{q} \mathcal{G}^S_{hs_{i_s}}$$

where

$$\mathcal{G}^S_{hs_{i_s}} = \mathcal{G}^S_{hs_i} \cup \{(\_{:}\mathtt{hs}_i\ \mathtt{qb4so{:}pcTopoRel}\ id^S(topoRel))\}$$

Example 6. The triples below show how the hierarchy steps of the Geography spatial hierarchy in the Customer dimension of the GeoNorthwind DW (Fig. C.7) are represented in RDF using Def. 9 and Ext. 9. Note that all hierarchy steps are spatial and have an associated topological relation.

```
# Hierarchy steps
_:customerGeography_hs1 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:customer ;
   qb4o:parentLevel gnw:city ;
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so:Within .
_:customerGeography_hs2 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:city ;
   qb4o:parentLevel gnw:state ;
   qb4o:pcCardinality qb4o:ManyToOne ;
```

```
   qb4so:pcTopoRel qb4so:Within .
_:customerGeography_hs3 a qb4o:HierarchyStep ;
   qb4o:inHierarchy gnw:customerGeography ;
   qb4o:childLevel gnw:state ;
   qb4o:parentLevel gnw:country ;
   qb4o:pcCardinality qb4o:ManyToOne ;
   qb4so:pcTopoRel qb4so:Within .
```

Definition 10. (Partial order on levels) The hierarchy steps $HS(h)$ of a hierarchy h define a *partial order* on the levels $l \in L(h)$. The reflexive and transitive closure of the partial order is denoted as $\sqsubseteq$, with a unique base level (l_b) and a unique top level (All), where all levels l are such that $l_b \sqsubseteq l$, and $l \sqsubseteq All$.

Definition 11. (Measures) An n-dimensional cube schema has a set of measures $M = \{m_1, \ldots, m_r\}$, which record the values of a phenomena being observed. Each measure $m_i \in M$ is defined in the cube schema graph $\mathcal{G}^S$ with the `qb:MeasureProperty` predicate. Similarly to attributes, each measure m_i is defined as ranging over XSD literals $\mathcal{L}$ with the `rdfs:range` property. The RDF graph formulation of the measures M is represented as

$$\mathcal{G}^S_M = \bigcup_{i=1}^{r} \mathcal{G}^S_{m_i}$$

where

$$\begin{aligned}\mathcal{G}^S_{m_i} = \\ &\{(id^S(m_i)\ \texttt{rdf:type qb:MeasureProperty})\} \cup \\ &\{(id^S(m_i)\ \texttt{rdfs:range}\ \mathcal{L})\}\end{aligned}$$

Extension 11. (Spatial measures) A measure is spatial if it is defined over a spatial domain as in spatial attributes (Ext. 8). A spatial measure m_{i_s} belongs to the set of spatial measures M_s, which is a subset of the set of measures M, such that $m_{i_s} \in M_s \subseteq M$. The RDF formulation of the spatial measures is similar as in Def. 11. However, the domain should range over spatial literals $\mathcal{L}_s$. The RDF graph formulation of the spatial measures M_s (w.r.t. Def. 11) is represented as

$$\mathcal{G}^S_{M_s} = \bigcup_{i=1}^{r} \mathcal{G}^S_{m_{i_s}}$$

where

$$\begin{aligned}\mathcal{G}^S_{m_{i_s}} = \\ &\{(id^S(m_{i_s})\ \texttt{rdf:type qb:MeasureProperty})\} \cup \\ &\{(id^S(m_{i_s})\ \texttt{rdfs:range}\ \mathcal{L}_s)\} \cup \{(id^S(m_{i_s})\ \\ &\texttt{rdfs:subClassOf geo:SpatialObject})\}\end{aligned}$$

Example 7. The triples below show how the measures of the GeoNorthwind DW (Fig. C.7) are represented in RDF using Def. 11 and Ext. 11. Note that SalesPoint is a spatial measure, which records the location of the stores in which the sales occurred. It is defined over Geometry domain as a Point type with WKT literal.

```
# Measures
gnw:quantity rdf:type qb:MeasureProperty ;
   rdfs:range xsd:integer .
gnw:unitPrice rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:discount rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:salesAmount rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:freight rdf:type qb:MeasureProperty ;
   rdfs:range xsd:decimal .
gnw:salesPoint rdf:type qb:MeasureProperty ;
   rdfs:domain geo:Point;
   rdfs:range geo:wktLiteral ;
   rdfs:subClassOf geo:SpatialObject .
```

Definition 12. (Fact) In an n-dimensional cube schema $\mathcal{CS} = (D, M, F)$, the fact F defines the structure of a cube with the `qb:DataStructureDefinition` property. The dimensions are given as `components` of the fact and are defined with the `qb4o:level` property. We assume that the fact F links the dimensions at the lowest granularity level, therefore `qb4o:level` links the lowest (base) level l_b of each dimension d_i, which is denoted as $l_b(d_i)$. The cardinality *card* of the relationship between a dimension level and a fact is represented with the `qb4o:cardinality` property. Similarly, the measures are given as `components` of the fact and are defined with the `qb:measure` property. The aggregate function *aggr* associated to each measure is represented with the `qb4o:aggregateFunction` property. The RDF graph formulation of the fact F is given in the following equation.

$$
\begin{aligned}
\mathcal{G}_F^S = \{&(id^S(F) \\
&\texttt{rdf:type qb:DataStructureDefinition})\} \cup \\
&\bigcup_{d_i \in D} \{(id^S(F) \ \texttt{qb:component} \\
&[\texttt{qb4o:level}\ id^S(l_b(d_i)); \\
&\texttt{qb4o:cardinality}\ id^S(card)])\} \cup \\
&\bigcup_{m_i \in M} \{(id^S(F) \ \texttt{qb:component} \\
&[\texttt{qb:measure}\ id^S(m_i); \\
&\texttt{qb4o:aggregateFunction}\ id^S(aggr)])\}
\end{aligned}
$$

Extension 12. (Spatial fact) A fact is spatial if it relates several levels, where two or more are spatial. A spatial fact may also have a topological relation *topoRel* that must be satisfied by the related spatial levels, which is represented with `qb4so:topologicalRelation`. This object property allows to specify a topological relation in fact-level relationship of spatial facts. The RDF graph formulation of such a fact is simply by adding the property of fact-level topological relation consecutively to the cardinality property as given in the following equation.

$$
\begin{aligned}
\mathcal{G}^S_{F_s} = \{ & (id^S(F_s) \\
& \texttt{rdf:type qb:DataStructureDefinition})\} \cup \\
& \bigcup_{d_i \in D} \{(id^S(F_s)\ \texttt{qb:component} \\
& [\texttt{qb4o:level}\ id^S(l_b(d_i)); \\
& \texttt{qb4o:cardinality}\ id^S(card); \\
& \texttt{qb4so:topologicalRelation}\ id^S(topoRel)]\} \cup \\
& \bigcup_{m_i \in M} \{(id^S(F)\ \texttt{qb:component} \\
& [\texttt{qb:measure}\ id^S(m_i); \\
& \texttt{qb4o:aggregateFunction}\ id^S(aggr)]) \}
\end{aligned}
$$

Example 8. The triples below show how the fact of the GeoNorthwind DW (Fig. C.7) is represented in RDF using Def. 12. Sales fact does not impose any topological relation between its spatial dimensions Supplier and Customer. SalesPoint is a spatial measure, which has a spatial aggregate function (AggrConvexHull).

```
# Cube definition
gnw:GeoNorthwind rdf:type
qb:DataStructureDefinition ;
   # Lowest level for each dimension
   qb:component [qb4o:level gnw:customer ;
   qb4o:cardinality qb4o:ManyToOne] ;
   qb:component [qb4o:level gnw:supplier ;
   qb4o:cardinality qb4o:ManyToOne] ;
   qb:component [qb4o:level gnw:product ;
   qb4o:cardinality qb4o:ManyToOne] ;
   ...
   # Cube measures
   qb:component [qb:measure gnw:quantity ;
      qb4o:aggregateFunction qb4o:Sum] ;
   qb:component [qb:measure gnw:unitPrice ;
```

```
   qb4o:aggregateFunction qb4o:Avg] ;
qb:component [qb:measure gnw:discount ;
   qb4o:aggregateFunction qb4o:Avg] ;
...
qb:component [qb:measure gnw:salesPoint ;
   qb4o:aggregateFunction qb4so:AggrConvexHull] .
```

4.2 Defining spatial data cube members with QB4SOLAP

We have explained in Sect. 4.1 how a data cube schema can be represented in RDF with QB4SOLAP. We show next how to use this schema to represent the instances of the GeoNorthwind DW (Fig. C.7) in RDF. We denote by $\mathcal{G}^I$ the RDF graph of the data cube instances. In the examples, we prefix the elements of $\mathcal{G}^I$ with `gnwi:`. The subgraph of $\mathcal{G}^I$ that refers to a specific cube *instance* x is denoted by $\mathcal{G}^I_x$ and the unique identifier of x is denoted by $id^I(x)$.

Definition 13. (Level members) A level l has a set of level members $LM(l) = \{lm_1, \ldots, lm_y\}$. Each level member lm_i has a unique IRI $id^I(lm_i) \in \mathcal{I}$, which is linked in the cube instance graph $\mathcal{G}^I$ with the `qb4o:LevelMember` predicate. A level member is related to its level by the `qb4o:memberOf` property. The RDF graph formulation of the level members $LM(l)$ is represented as

$$\mathcal{G}^I_{LM(l)} = \bigcup_{i=1}^{y} \mathcal{G}^I_{lm_i}$$

where

$$\begin{aligned}\mathcal{G}^I_{lm_i} = \\ &\{(id^I(lm_i)\ \texttt{rdf:type}\ \texttt{qb4o:LevelMember})\}\cup \\ &\{(id^I(lm_i)\ \texttt{qb4o:memberOf}\ id^S(l))\}\end{aligned}$$

Definition 14. (Attributes of level members) A level member lm has a set of attributes $A(lm) = \{a_1, \ldots, a_p\}$, which are used to describe the characteristics of the level member (Def. 8). Each attribute a_i is linked to the level member with the identifier $id^S(a_i)$. We denote by $lm \rightsquigarrow v_{a_i}$ the value v_{a_i} that a level member lm associates to attribute a_i. This value is given as a literal $\mathcal{L}$ such that $v_{a_i} \in \mathcal{L}$. The RDF graph formulation of the attributes $A(lm)$ is represented as

$$\mathcal{G}^I_{A(lm)} = \bigcup_{i=1}^{p} \mathcal{G}^I_{a_i}$$

where

$$\mathcal{G}^I_{a_i} = \{(id^I(lm)\ \ id^S(a_i)\ \ v_{a_i}) \mid lm \rightsquigarrow v_{a_i}\}$$

Definition 15. (Partial order on level members) A hierarchy step $hs = (l_c, l_p, card)$ between a child level l_c and a parent level l_p defines a set of roll-up relations $RU(hs) = \{r_1, \ldots, r_k\}$ where each $r_i = lm_{c_i} \sqsubseteq lm_{p_i}$ relates a child level member $lm_{c_i} \in LM(l_c)$ to a parent level member $lm_{p_i} \in LM(l_p)$. These roll-up relations define a *partial order* between level members with regard to Def. 10 and are expressed using the property `skos:broader`. The RDF graph formulation of the roll-up relations $RU(hs)$ is represented as

$$\mathcal{G}^I_{RU(hs)} = \bigcup_{i=1}^{k} \mathcal{G}^I_{r_i}$$

where

$$\mathcal{G}^I_{r_i} = \{(id^I(lm_c) \texttt{ skos:broader } id^I(lm_p)) \mid r_i = lm_{c_i} \sqsubseteq lm_{p_i}\}$$

Example 9. The triples below show how some level members of the GeoNorthwind DW (Fig. C.7) are represented in RDF using Defs. 13–14.

```
gnwi:customer_1 rdf:type qb4o:LevelMember ;
   qb4o:memberOf gnw:customer ;
   gnw:customerID 1 ;
   gnw:customerName "Alfreds Futterkiste" ;
   gnw:address "Obere Str.  57" ;
   gnw:postalCode "12209" ;
   gnw:customerGeo
   "POINT(13.099 52.401)"^^geo:wktLiteral ;
   skos:broader gnwi:city_6 .
gnwi:city_6 rdf:type qb4o:LevelMember ;
   qb4o:memberOf gnw:city ;
   gnw:cityName "Berlin" ;
   gnw:cityGeo
   "POINT(13.4060 52.519)"^^geo:wktLiteral ;
   skos:broader gnwi:state_224 .
```

Definition 16. (Fact members) A fact F has a set of fact members $FM(F) = \{f_1, \ldots, f_t\}$, which are the instances of the data cube. Each fact $f_i \in FM$ has a unique IRI $id^I(f_i) \in \mathcal{I}$, which is linked in the cube instance graph $\mathcal{G}^I$ with the `qb:Observation` predicate.

A fact member f_i is related to a set of dimension levels $L(f_i) = \{l_1, \ldots, l_r\}$ and has a set of measures $M(f_i) = \{m_1, \ldots, m_s\}$. Each dimension level l_j is linked to the level member with the identifier $id^S(l_j)$ and each measure m_k is linked to the level member with the identifier $id^S(m_k)$. We denote by $f \rightsquigarrow v_{l_j}$ and $f \rightsquigarrow v_{m_k}$, respectively, the dimension values and measure values associated with a fact f. The value $v_{l_j} \in \mathcal{I}$ is the identifier of a level

member in $LM(l_j)$. Further, the value v_{m_k} for every measure m_k is a literal such that $v_{m_k} \in \mathcal{L}$. The RDF graph formulation of the fact members $FM(F)$ is represented as

$$\mathcal{G}^I_{FM(F)} = \bigcup_{i=1}^{t} \mathcal{G}^I_{f_i}$$

where

$$\begin{aligned}\mathcal{G}^I_{f_i} = \{&(id^I(f_i)\ \texttt{rdf:type qb:Observation})\}\cup \\ &\bigcup_{l_j \in L(f_i)} \{(id^I(f_i)\ \ id^S(l_j)\ \ id^I(v_{l_j}) \mid f_i \rightsquigarrow v_{l_j})\}\cup \\ &\bigcup_{m_k \in M(f_i)} \{(id^I(f_i)\ \ id^S(m_k)\ \ id^I(v_{m_k}) \mid f_i \rightsquigarrow v_{m_k})\}\end{aligned}$$

Example 10. The triples below show how a fact member of the GeoNorthwind DW (Fig. C.7) is represented in RDF using Defs. 12–16. Note that the fact member and corresponding level members relating to dimensions are given with the prefix `gnwi:`. $id^S(a_{ID})$ is the surrogate key (Def. 8) that links the fact member to the corresponding dimensions' base level members.

```
gnwi:sale_10613_1 rdf:type qb:Observation ;
   gnw:customer gnwi:customer_1 ;
   gnw:supplier gnwi:supplier_6 ;
   gnw:product gnwi:product_13 ;
   ...
   gnw:quantity 8 ;
   gnw:unitPrice "6,00"^^xsd:decimal ;
   gnw:discount "0,10"^^xsd:decimal ;
   gnw:salesPoint
   "POINT(23.08 42.34)"^^geo:wktLiteral.
```

5 Semantics of SOLAP operators

This section defines a formal algebra for SOLAP operators. Examples of the operators are provided after their definitions. The complete SPARQL query examples are given at our website[13] and can be tested at our public endpoint[14]. The query runtimes for each SOLAP operator are given in Appendix A1, Table C.5 for the use case dataset GeoNorthwind (Sect. 4.1, Fig. C.7). These operators can be applied on spatially enhanced multidimensional data

[13] http://extbi.cs.aau.dk/SOLAP4SW/queries
[14] http://extbi.lab.aau.dk/sparql

cubes (Sect. 3.3). The presentation defines the semantics of a SOLAP operator by logically specifying the typical OLAP operators with spatial functions and conditions. Spatial functions and conditions can be selected from a range of operation classes, which can be applied on spatial data types (Sect. 3.1). Let $\mathcal{S}$ be the set of any spatial operators where $\mathcal{S} = (\mathcal{S}_{agg} \cup \mathcal{T}_{rel} \cup \mathcal{N}_{op})$, used to represent a *spatial predicate* $\phi^{\mathcal{S}} \in \mathcal{S}$ or a *spatial function* $\mathtt{f}^{\mathcal{S}} \in \mathcal{S}$, which is in a SOLAP operator. The following SOLAP operators are defined with a spatial extension to the well-known OLAP operators, which are given in the remarks.

Remark 17. (Slice) The *slice* operator removes a dimension from a cube $\mathcal{C}$ by selecting one instance in a dimension level. For example, the query "slice on customers in the city of Odense" is a slice operation. (Cube is the sales, dimension is the customer, level in dimension is the city and the value is Odense, which is sliced out from the cube).

Definition 17. (S-Slice) The *s-slice* operator removes a dimension from a cube $\mathcal{C}$ by choosing a single spatial attribute value $v_s \in \mathcal{L}_s$ (Ext. 8) in a spatial level l_s (Ext. 7).

As for the semantics, s-slice takes an n-dimensional cube $\mathcal{C}$ as an argument. We assume that the cube has the cube schema $\mathcal{CS} = (D, M, F)$, with the fact members $f \in FM$ as given in Def. 16. As parameters, s-slice takes a spatial literal value v_s, the base level l_b and the target (spatial) level l_s of a dimension d_i. The base level l_b specifies the dimension d_i (Def. 16). The target spatial level l_s is the level, that the spatial literal value v_s is related.

The operator is defined as: $\mathcal{SS}(\mathcal{C})[l_b, l_s, v_s] = \mathcal{C}'$, which returns a cube $\mathcal{C}'$ with $n-1$ dimensions and the schema $\mathcal{CS} = (D', M', F')$, where $D' = D \setminus \{d_i\}$, $M' = M$, and $F' = F$. The measures M and the fact type F remains the same though the new cube $\mathcal{C}'$ has one dimension less.

The s-slice operator selects a subset FM' from the set of fact members FM ($FM' \subseteq FM$), with respect to the given parameter v_s. Assuming that the granularity of the fact members are at the (lowest) base level of the dimension $l_b \in L(d_i)$ in the given cube, a partial order exists among the levels, from bottom level to the target spatial level l_s such that $l_b \sqsubseteq l_s$. The given parameter v_s is related to a level member of the level l_s. We say that the fact members are characterized by dimension values, which is written as $f \rightsquigarrow v_{d_i}$ where $v_{d_i} \equiv v_{l_{b(d_i)}}$ (Def. 16). In other words, dimensions are associated to the fact members by the values of the dimensions' base level members $v_{l_{b(d_i)}}$. When the dimension d_i is clear in the context, we will use base level v_{l_b} for simplicity reasons.

To sum up, the subset FM' of facts is selected with regards to the partial order on levels from base level l_b to the target level l_s. The value v_{l_s} in the target level l_s is specified with respect to the given spatial literal value

v_s. The value of v_s might be equal to a spatial attribute value in the target level l_s, thus v_{l_s} is characterized by the attribute value v_s and written as $v_{l_s} \rightsquigarrow v_s$ (Ex. 11). Or, v_s is an arbitrary spatial literal that entails a topological relation $\mathcal{T}_{rel}$ (i.e., within) in a value of the target spatial level v_{l_s}, which is written as $\exists v_{l_s} : \phi^{\mathcal{S}}(v_s)$ where $\phi^{\mathcal{S}}$ is a *spatial Boolean predicate* that represents a topological relation (Ex. 12). After applying the s-slice operator on cube $\mathcal{C}$, the new (sub)set of fact members is defined for both cases respectively as follows; $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge v_{l_s} \rightsquigarrow v_s\}$, $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} : \phi^{\mathcal{S}}(v_s)\}$.

Example 11. With regards to the traditional slice query "slice on customers in the city of Odense", in *s-slice*, the user could specify a geometry extent (e.g., polygon coordinates of the city of Odense) as spatial literal for slicing instead of giving a text literal (e.g., "Odense"). So the s-slice query would be; "slice on customers of the city, which has the geometry `"POLYGON((10.43951 55.47006, 10.439472 55.470036, 10.439240(...))"`". More intuitively, instead of the specified spatial literal $v_s \in \mathcal{L}_s$, the user can pass a function call as parameter to s-slice, e.g., by querying "slice on customers in the largest city of (southern) Denmark by land area". The function call should calculate the area of the cities by their geometries where the largest city is selected as a requirement of the s-slice operator. Both cases are given in the following.

Example 11.1. The following SPARQL query shows an s-slice operator, which filters with the given spatial literal by the user.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city .
   ?city gnw:cityGeo ?cityGeo .
FILTER (?cityGeo = "POLYGON((10.439517 55.470064,
10.4394729 55.4700361, 10.4392403 (...))") }
```

Example 11.2. The following SPARQL query shows the s-slice operator, which filters with the function call (largest city) returned from inner select. Given the current limitations of SPARQL, there is not an area calculation function from the geometries of the spatial objects during query run time, however we give the query with a notional built-in `bif:st_area` function.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
```

```
      skos:broader ?city .
   ?city gnw:cityGeo ?cityGeo .
# Inner select for finding the largest city
   {SELECT ?x (MAX(?area) as ?maxArea)
   WHERE {
      ?obs rdf:type qb:Observation ;
         gnw:customerID ?cust .
      ?cust qb4o:memberOf gnw:customer ;
         skos:broader ?city .
      ?city gnw:cityGeo ?x .
   BIND(bif:st_area(?x) as ?area)}
FILTER (?cityGeo = ?x) }
```

Example 12. With regards to the traditional slice query "slice on customers in the city of Odense", in this example of *s-slice*, the user gives a point geometry (i.e., X, Y coordinates of a point as spatial literal) and filter at the given level (i.e., City level) that the given point is within. So the s-slice query would be; "slice on customers of the city, in which the given `"POINT(10.43951 55.47006)"` is within".

The following SPARQL query shows an s-slice operator, which filters at the specified level with the given spatial literal by the user.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city .
   ?city gnw:cityGeo ?cityGeo .
FILTER (bif:st_within("POINT(10.43951 55.47006)",
?cityGeo)) }
```

Note that the s-slice can be operated in different ways based on the geometry given to the query. In both Ex.s 11 and 12, slice level is given as City, however in Ex. 12 a random X, Y point is given that is falling into the target city. Therefore we need to use *within* from topological relationships ($\mathcal{T}_{rel}$) class in order to verify and filter that city.

Remark 18. (Dice) The traditional *dice* operator takes a cube and a Boolean condition ϕ, which returns a new cube containing only the cells that satisfy the Boolean condition ϕ. Dice operation is analogous to relational algebra, *R selection*; $\sigma_{\phi}(R)$, but the argument is a cube not a relation. For example, the query "sales to customers of type LLC (Limited Liability Company)" is a dice operation. (Cube is the sales, dimension is the customer, and Boolean condition is the customer type if they are LLC).

Definition 18. (S-Dice) Similarly, the *s-dice* operator takes an n-dimensional cube $\mathcal{C}$ as an argument, which has the cube schema $\mathcal{CS} = (D, M, F)$ with the fact members $f \in FM$ as given in Def. 16. As a parameter s-dice takes a spatial Boolean predicate, which is denoted by $\phi^{\mathcal{S}}$. The s-dice operator keeps the cells of the cube $\mathcal{C}$ that satisfies the spatial predicate over spatial dimension levels l_s, attributes a_s, and measures m.

The semantics of the operator is defined as:
$\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}] = \mathcal{C}'$ where spatial predicate $\phi^{\mathcal{S}}$ can be applied on spatial level member values $\phi^{\mathcal{S}}(v_{l_s})$, spatial attribute values $\phi^{\mathcal{S}}(v_{a_s})$, and measure values $\phi^{\mathcal{S}}(v_m)$ or a combination of these.

$\mathcal{SD}$ operator returns a sub cube $\mathcal{C}' \subseteq \mathcal{C}$, which has the schema $\mathcal{CS} = (D', M', F')$ where $D' = D$, $M' = M$, and $F' = F$. Unlike the s-slice operator, s-dice keeps all the dimensions D in the output cube $\mathcal{C}'$. The set of measures M and the fact type F also remains the same, though the new cube $\mathcal{C}'$ is a subset of the original cube $\mathcal{C}$ with filtered fact members $f \in FM'$, which is explained in the following.

The s-dice operator selects a subset FM' of the fact members' set $FM' \subseteq FM$ with respect to the spatial predicate $\phi^{\mathcal{S}}$ on level members as follows;

1. Spatial predicate on level values: $FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge \phi^{\mathcal{S}}(v_{l_s})\}$.
2. Spatial predicate on level attribute values:
$FM' = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b), v_{l_s} \in LM(l_s) \wedge v_{l_s} \rightsquigarrow v_{a_s} : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq v_{l_s} \wedge v_{l_s} \rightsquigarrow v_{a_s} \wedge \phi^{\mathcal{S}}(v_{a_s})\}$.
Note that the filtering the facts through level members can be done by v_{l_s} (level values) or attribute values v_{a_s} by applying the spatial predicate $\phi^{\mathcal{S}}$. Finally filtering of the facts is on associated measure values is defined in the following;
3. Spatial predicate on measure values of m_s: $FM' = \{f \in FM \mid \exists\, v_{m_s} \in Codomain(m_s) : f \rightsquigarrow v_{m_s} \wedge \phi^{\mathcal{S}}(v_{m_s})\}$.
For complex cases, i.e., combining these three types; the result set is also followed by combining the basic result sets.

Example 13. The s-dice operator can be implemented on level and attribute values by filtering level members in the cube or on measures by filtering the facts in the cube. In both cases the spatial predicate $\phi^{\mathcal{S}}$ is used.

The query for the s-dice operator could be "sales to customers, which are located within 5 km distance from their city center" where the s-dice is on level members by filtering the customer level. The spatial predicate $\phi^{\mathcal{S}}$ can be interpreted in two different ways (See Appendix A1 for comparison of their query run times).

Example 13.1. First method is assuming a buffer area of 5 km from the coordinates of city center and checking customers' locations by *within* operator

from topological relations $\phi^{\mathcal{S}} \in \mathcal{T}_{rel}$ if it meets the condition. The following SPARQL query shows the implementation of this method on level members.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city ;
      gnw:customerGeo ?custGeo .
   ?city gnw:cityGeo ?cityCentGeo .
FILTER (bif:st_within (?custGeo, ?cityCentGeo,
5))}
```

Example 13.2. Second method is checking if the distance from a customer location to the corresponding city center is less than 5 km, by using *distance* function from numeric operations $\mathtt{f}^{\mathcal{S}} \in \mathcal{N}_{op}$. In this case the spatial predicate $\phi^{\mathcal{S}}$ is a combination of a spatial function $\mathtt{f}^{\mathcal{S}}$ and a regular Boolean predicate ϕ. Spatial function is *distance* from numeric operations and the predicate is *less than* ($<$). The following SPARQL query shows the implementation of this method for s-dice on level members.

```
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city ;
      gnw:customerGeo ?custGeo .
   ?city gnw:cityGeo ?cityCentGeo .
BIND (bif:st_distance (?custGeo, ?cityCentGeo)
AS ?distance) FILTER ( ?distance < 5 ) }
```

Remark 19. (Roll-up) The traditional *roll-up* operator aggregates measures according to a dimension hierarchy (by using an aggregate function), in order to obtain measures at a coarser granularity for a given dimension. For example, the query "total amount of sales to customers by city" is a classical roll-up operation. (Cube is the sales, dimension is the customer, level in dimension to roll-up is the city such that *customer* $\sqsubseteq$ *city*, measure is the sales amount and aggregate function is the sum in order to calculate the total sales.)

Definition 19. (S-Roll-up) Similarly to roll-up operator, *s-roll-up* aggregates measures $m \in M$ of a given cube $\mathcal{C}$, by using an aggregate function and a spatial function $\mathtt{f}^{\mathcal{S}} \in \mathcal{S}$ (Sect. 3.1) along a spatial dimension's hierarchy h_s (Ext. 6), which should have spatial levels l_s (Ext. 7). However, in s-roll-up the dimension hierarchy is created dynamically on levels by the spatial function $\mathtt{f}^{\mathcal{S}}$. We call this hierarchy a *dynamic spatial hierarchy*, conceptually from a

base level l_b to the dynamically created target level l'_s such that $l_b \sqsubseteq_d l'_s$. The instances of the target level l'_s are obtained by the spatial function $\mathtt{f}^{\mathcal{S}}(l_s)$ that is applied on spatial dimension levels.

As for the semantics, s-roll-up takes an n-dimensional cube $\mathcal{C}$ as an argument, which has the cube schema $\mathcal{CS} = (D, M, F)$ with the fact members $f \in FM$ as given in Def. 16. As a parameter s-roll-up takes a spatial function $\mathtt{f}^{\mathcal{S}} \in \mathcal{S}$ to operate on levels $L(d_i)$ and an aggregate function agg to calculate a measure m at the higher target level. For simplicity of explanation and without loss of generality, we initially assume that there is only one measure m. The extension of the operator on several measures $m \in M$ is explained in the last paragraph. S-Roll-up operator is formulated as; $\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)] = \mathcal{C}'$, which returns a cube $\mathcal{C}'$ with n-dimensions and has the schema $\mathcal{CS} = (D', M', F')$ where $F' = F$, $M' = M$, and $D' = \{d_i \in D \mid \{d_1, \ldots, d'_i, \ldots, d_n\} \wedge L'(d'_i) = L(d_i) \setminus (l_b \sqsubseteq_d \ldots \sqsubset_d l_s) \cup \{l'_s\}\}$. After the s-roll-up operation, number of dimensions in D remains the same, although the base levels and levels below the target level $(l_b \sqsubseteq_d \ldots \sqsubset_d l_s)$ of the corresponding dimension d_i are left out and a new target level l'_s is added to the set of dimension levels $L'(d'_i)$ of d'_i.

The set of level members of the level l'_s is selected with respect to the spatial function on base level members of a spatial dimension such that $LM(l'_s) = \{\mathtt{f}^{\mathcal{S}}(v_{l_b}) \mid v_{l_b} \in LM(l_b)\}$ where $l_b \sqsubseteq_d l'_s \iff \mathtt{f}^{\mathcal{S}}(v_{l_b}) = v_{l'_s}$, which means that the base level l_b rolls up along the spatial dynamic hierarchy ($\sqsubseteq_d$) to the target new spatial level l'_s if and only if spatial function on base level $\mathtt{f}^{\mathcal{S}}(v_{l_b}) = v_{l'_s}$ produces the new spatial level members $v_{l'_s}$. Even though the set of measures M remains the same, the s-roll-up operator obtains the measure values associated with fact members f' at a coarser granularity l'_s, which alters the set of facts $FM' \nsubseteq FM$. In order to create the new set of facts FM' at the new granularity level l'_s, the $Group$ operator [30] is used to group the facts characterized by the same level members $v_{l'_s} \in LM(l'_s)$ such that $Group(v_{l'_s}) = \{f \in FM \mid \exists\, v_{l_b} \in LM(l_b) : f \rightsquigarrow v_{l_b} \wedge v_{l_b} \sqsubseteq_d v_{l'_s}\}$. The output of the $Group$ operator on level members is a new fact instance f'. In order to aggregate the measure values v_m, which are associated with the fact members f we use an aggregate function agg such that $agg(\{f_1, \ldots, f_k\}) = agg(v_{m_1}, \ldots, v_{m_k})$ where $f_i \rightsquigarrow v_{m_i}$, $i = 1, \ldots, k$. Finally, the set of the new facts $f' \in FM'$ is constructed, that is given with the associated new level members and aggregated measure values as; $FM' = \{f' = Group(v_{l'_s}) \mid \exists\, v_{l'_s} \in LM(l'_s) : f' \rightsquigarrow v_{l'_s} \wedge f' \rightsquigarrow agg(Group(v_{l'_s}))\}$.

The extension to multiple measures is similar, which is done by providing and using a separate aggregate function for each measure $m \in M$.

Example 14. The following SPARQL query shows the s-roll-up operator, which is exemplified in Sect. 3.6. The query is "total amount of sales to customers by city of the closest suppliers". Note that the measures are aggregated up to a

new city from customer level of the customer dimension, which is specified as the *Closest City*. The hierarchy step from customer to city is defined dynamically by a spatial function $\mathtt{f}^{\mathcal{S}}$ (*distance* from numeric operations $\mathcal{N}_{op} \subset \mathcal{S}$), which is then used in a wrapper function to find the closest distance of the suppliers and customers. The levels and level members (of customer), which are below the newly defined level (Closest City) are left out in the result.

```
SELECT ?city (SUM(?sales) AS ?totalSales)
WHERE { ?obs rdf:type qb:Observation ;
   gnw:customerID ?cust ;
   gnw:supplierID ?sup;gnw:salesAmount ?sales .
?cust qb4o:memberOf gnw:customer ;
   gnw:customerGeo ?custGeo ;
   gnw:customerName ?custName ;
   skos:broader ?city .
?city qb4o:memberOf gnw:city .
?sup gnw:supplierGeo ?supGeo .
# Inner Select for the total sales to
# the closest supplier of the customer
   { SELECT ?cust1 (MIN(?distance) AS
   ?minDistance) WHERE
   { ?obs rdf:type qb:Observation ;
      gnw:customerID ?cust1 ;
      gnw:supplierID ?sup1 .
   ?sup1 gnw:supplierGeo ?sup1Geo .
   ?cust1 gnw:customerGeo ?cust1Geo .
   BIND (bif:st_distance( ?cust1Geo, ?sup1Geo )
   AS ?distance)}
   GROUP BY ?cust1 }
   FILTER (?cust = ?cust1 && bif:st_distance
   (?custGeo, ?supGeo) = ?minDistance)}
GROUP BY ?city
```

Remark 20. (Drill-down) Drill-down is the inverse operator of roll-up, which disaggregates previously summarized data to a child level in order to obtain measures at a finer granularity of a given dimension. For example, the roll-up query given in Remark 19 ("total amount of sales to customers by city") aggregates sales by summing up the sales amount, from customer level to city level along a hierarchy. As drill-down operator performs the operation opposite to the roll-up an example would be; "average amount of sales of each supplier, drilled down from the city level to the supplier level". (Cube is the same as sales, and the hierarchy is the same but the dimension is the supplier, so child level in dimension to drill-down from city level is the supplier such that $City \sqsupseteq Supplier$). Conceptually, a drill-down to level l_i on a cube $\mathcal{C}$ corresponds to a roll-up to the same level l_i on the *base cube* of $\mathcal{C}$, that is denoted as $BaseCube(\mathcal{C})$.

Definition 20. (S-Drill-down) Analogously to drill-down operator, *s-drill-down* disaggregates measures $m \in M$ of a given cube $\mathcal{C}$, by using an aggregate function and a spatial function $\mathtt{f}^{\mathcal{S}}$ (Sect. 3.1) along a spatial dimension's hierarchy h_s (Ext. 6), which should have spatial levels (i.e., l_s) (Ext. 7).

Conceptually, in s-drill-down, the dimension hierarchy is created dynamically on levels by the spatial function $\mathtt{f}^{\mathcal{S}}$ as in s-roll-up. This is similar to the dynamic spatial hierarchy defined in Def. 19, that is from a spatial parent level l_{p_s} to a dynamically created spatial child level l'_{c_s} such that $l_{p_s} \sqsupseteq_d l'_{c_s}$. The target spatial child level l'_{c_s} is the output of the spatial function $\mathtt{f}^{\mathcal{S}}$ on spatial levels $l_{i_s} \in L(d_i)$ of the spatial dimension. Applying s-drill-down to child level l'_{c_s} from a parent level l_{p_s} on a cube $\mathcal{C}$ corresponds to applying s-roll-up to the same level l'_{c_s} from the base level l_b on the *base cube* of $\mathcal{C}$. Therefore, the semantics of the s-drill-down is described same as s-roll-up and the operator is formulated as $SDD(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)] = SRU(BaseCube(\mathcal{C}))[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)]$.

Example 15. In order to exemplify an s-drill-down, starting from the result cube graph of Ex. 14 ("total amount of sales to customers by city of the closest supplier"), which is at the granularity of City level, we drill down to child level Supplier with the query "average amount of sales of furthest suppliers to their city center, drilled down the from City level to Supplier level". The following SPARQL query shows the given example.

```
SELECT ?sup (AVG(?sales) AS ?averageSales)
   (MAX(?distance) AS ?maxDistance)
WHERE { ?obs rdf:type qb:Observation ;
   gnw:supplierID ?sup ;
   gnw:salesAmount ?sales .
?sup qb4o:memberOf gnw:supplier ;
   gnw:supplierGeo ?supGeo ;
   gnw:supplierName ?supName ;
   skos:broader ?city .
?city qb4o:memberOf gnw:city ;
   gnw:cityGeo ?cityCentGeo .
BIND (bif:st_distance(?supGeo, ?cityCentGeo)
   AS ?distance)}
FILTER (?distance = ?maxDistance)}
GROUP BY ?sup
```

In this paper, we focus on direct querying of single data cubes with main SOLAP operators in SPARQL. The integration of several cubes through *s-drill-across* or set-oriented operations such as *union, intersection, and difference* [31] is out of scope and remained as future work.

6 Generating SOLAP queries in SPARQL via QB4SOLAP

After having defined the high-level SOLAP operators in Sect. 5, this section first describes how to generate SPARQL queries for each of these operators by using the QB4SOLAP metamodel (Sect. 4). Afterwards, this section describes how to create more complex SPARQL queries for nested SOLAP operations.

6.1 Generation algorithms

The generated SPARQL queries Q are of the form "$Q =$ `SELECT` R `WHERE` GP", where GP is a graph pattern containing triple patterns and R is the (set of) variable(s) that are returned in the result of the query. Triple patterns are based on triples of the form (s, p, o) (Def. 4), where triple components are replaced by variables. A set of triple patterns defines a graph pattern GP. Given an RDF graph $\mathcal{G}$, a graph pattern GP is used to search for subgraphs $\mathcal{G}_{(R)} \subseteq \mathcal{G}$ matching the pattern. In our algorithm, the graph pattern is initially empty, $GP = \emptyset$, and the triple patterns are added incrementally to the body of the `WHERE` clause: $GP = GP \cup (s\ \ p\ \ o)$.

RDF datasets published with the QB4SOLAP vocabulary use the predicate `skos:broader`, to define the roll-up relation from child level to parent level (Defs. 13 and 15). As this is the case for all hierarchy levels in a dimension, every OLAP query contains such roll-up paths that we need to consider as part of GP in the `WHERE` clause.

Thus, we define a helper function *RUPath* (Algorithm 1) that we can use in the SOLAP query generation algorithms.

Algorithm 1: $RUPath(\mathcal{G}^S_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?a_s, ?f) : GP$

Input: $\mathcal{G}^S_{(\mathcal{C})}$, l_b, l_s, a_{ID},?a_s, ?f
Output: GP

begin
  GP = (?f `rdf:type qb:Observation`)
  $GP = GP \cup$ (?f $id^S(a_{ID})$?l_b $\wedge$?l_b `qb4o:memberOf` $id^S(l_b)$)
  foreach $(id^S(l_c), id^S(l_p)) \in \mathcal{G}^S_{(\mathcal{C})} \mid l_p \sqsubseteq l_s$ **do**
    $GP = GP \cup$ (?l_b `skos:broader` ?l_p $\wedge$?l_p `skos:broader` ?l_s)
  let $GP = GP \cup$ (?l_s $id^S(a_s)$?a_s)
return GP

Build roll-up path (*RUPath*). The helper function *RUPath* returns a graph pattern that we can use in the body of the `WHERE` clause. The roll-up path

pattern is created as a path-shaped join of triples with partial order ($\sqsubseteq$, `skos:broader`) (Def.s 10 and 15). The triple pattern is of the form $\{(s_1\ \ p_1\ \ o_1), (o_1\ \ p_2\ \ o_2), (o_2\ \ p_2\ \ o_3), \ldots (o_{n-1}\ \ p_2\ \ o_n)\}$ where s_1 is the root of the graph pattern and corresponds to fact members f from the QB4SOLAP schema (Def. 16), p_1 is the predicate $id^{\mathcal{S}}(a_{ID})$ that associates facts with level members v_{l_i} ($f \rightsquigarrow v_{l_i}$, Def. 16), o_1 is the variable for the first level member that rolls up to its parent level o_2 such that $o_1 \sqsubseteq o_2$ and so on, and the p_2 predicate corresponds to the `skos:broader` property. The last variable in the path o_n corresponds to the target level l_s in order to represent the level member variables at the target level. The roll-up path starts at the fact instances f (Def. 16). Afterwards, the partial order on level members (Def. 15) from base level l_b to target level l_s is applied. Algorithm 1 sketches the helper function for building the roll-up path for dimensions; from facts to dimension levels with predicates and cube member IRIs defined in the cube schema.

In order to represent such varying parameters at the instance level such as fact members, level members, or parameter values given by the user, and to distinguish these parameters from other parameters in the algorithm, we represent such parameters using variable names with question marks.

We use a `FILTER` expression to restrict the output data by using a (spatial) Boolean predicate $\phi^{\mathcal{S}}$. A `FILTER` expression is part of the `WHERE` clause in a SPARQL query. Therefore, it is added to the body of the `WHERE` clause in the graph pattern GP as $GP = GP \cup (\texttt{FILTER}\ \phi^{\mathcal{S}})$. In the cases where there is a spatial function $\texttt{f}^{\mathcal{S}}(x)$ in the SOLAP operator, it is given in the `BIND` clause, which is technically a part of the `WHERE` clause and therefore added to the body of the `WHERE` clause in a graph pattern GP as $GP = GP \cup (\texttt{BIND}\ \texttt{f}^{\mathcal{S}}(x))$. SPARQL 1.1 defines aggregate expressions[15], such as `SUM, MIN, MAX, AVG`, etc.

We apply them on measure values or use them as wrappers in spatial functions. In the following, we often write `AGG` to represent them.

In the following, we present the SPARQL query generation algorithms for the SOLAP operators defined in Sect. 5. The algorithms take the input parameters and arguments of the SOLAP operator and return the a SPARQL query Q that can be executed.

S-Slice generator. To generate a SPARQL query for the s-slice operator $\mathcal{SS}(\mathcal{C})[l_b, l_s, v_s]$ (Def. 17), we use Algorithm 2. Parameter v_s is a spatial literal value $v_s \in \mathcal{L}_s$ (i.e., `POINT` or `POLYGON`) that should be related to a spatial level l_s (Ext. 7). This means that v_s is defined as a polygon geometry that corresponds to a spatial attribute value in the target level l_s (Ex. 11) or v_s is defined as a point geometry that is spatially contained in a spatial attribute value of the target level l_s (Ex. 12). Note that in Ex. 11.1, the given spatial

[15] `https://www.w3.org/TR/sparql11-query/#aggregates`

literal has the geometry data type polygon, which corresponds to a spatial level attribute a_s (Ext. 8) at a spatial level l_s. Similarly, the spatial function call $\mathtt{f}^{\mathcal{S}}(x)$ in Ex. 11.2 returns a polygon that corresponds to a spatial level attribute a_s.

Algorithm 2: *S–SliceGenerator* $(\mathcal{G}^{I}_{(\mathcal{C})}, v_s, l_b, l_s) : Q$

Input: $\mathcal{G}^{I}_{(\mathcal{C})}, v_s, l_b, l_s$
Output: Q
1 **begin**
2 $Q = \emptyset$; GP = RUPath($\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?$\mathrm{a}_s$,?f)
3 **if** v_s *is a* POINT **then**
4 $GP = GP \cup$ (FILTER(st_within v_s, ?a_s))
5 **else if** $v_s = \mathtt{f}^{\mathcal{S}}(x)$ **then**
6 $Q' = \emptyset$; GP' =RUPath($\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}$,?$\mathrm{a}_s$,?f)
7 $GP' = GP' \cup$ (BIND $\mathtt{f}^{\mathcal{S}}(x)$ AS ?v_x)
8 Q' = SELECT ?x AGG(?v_x) WHERE GP'
9 $GP = GP \cup Q' \cup$ (FILTER ?x = ?a_s)
10 **else**
11 $GP = GP \cup$ (FILTER v_s = ?a_s)
12 **return** Q = SELECT ?f WHERE GP

On the other hand the given spatial literal in Ex. 12 has the geometry data type point, which corresponds to the spatial level l_s via topological relations ($\mathcal{T}_{rel}$). We consider all these possibilities in the s-slice generator algorithm. We explained these in the following, where the steps are referencing the line numbers in Algorithm 2.

Line 2. Get the path for dimension d_s (e.g., Customer) from the observation facts f to the base level l_b, and build path-shaped triple pattern paths from the dimension's base level l_b to the target spatial level l_s (e.g., City level). Finally, get level attribute IRIs and variables for the spatial attributes a_s (e.g., City geometry). All this is done by the RUPath function (Algorithm 1) that is used by the s-slice generator. The following shows an example result of this step that is added to GP:

```
{?obs rdf:type qb:Observation ;
    gnw:customerID ?cust .
?cust qb4o:memberOf gnw:customer ;
    skos:broader ?city .
?city gnw:cityGeo ?cityGeo .
```

Line 3. Check if the spatial literal v_s is a point geometry type. If true, create a FILTER statement with a spatial Boolean predicate (Line 4) and go to the result (Line 12).

Line 4. Build the FILTER statement based on the spatial literal v_s and the spatial attribute a_s (Ex. 12). As a result the following lines might be added to the GP:

```
FILTER (bif:st_within("POINT(10.43951
55.47006)", ?cityGeo))}
```

Line 5. Check if v_s is a function call $\mathtt{f}^{\mathcal{S}}(x)$. If true (Ex. 11.2), construct an inner select query to compute the spatial function $\mathtt{f}^{\mathcal{S}}(x)$, then go to the result (Line 12).

Line 6. Call the `RUPath` function in order to link a_s variables with the fact instances (this time for inner select query Q'). This step creates a graph pattern GP' for inner select query Q', for example:

```
{?obs rdf:type qb:Observation ;
   gnw:customerID ?cust .
?cust qb4o:memberOf gnw:customer ;
   skos:broader ?city .
?city gnw:cityGeo ?x .
```

Line 7. Build a bind statement on a_s variables for calculating spatial functions (e.g., compute areas). For example, the following lines might be added to graph pattern GP':

```
BIND (bif:st_area(?x) as ?area) }
```

Line 8. Generate the inner select query Q' based on GP' generated in Lines 6 and 7. For example (Q' finds the geometry of the largest city):

```
Q' = {SELECT ?x (MAX(?area) as ?maxArea)
   WHERE {?obs rdf:type qb:Observation ;
      gnw:customerID ?cust .
   ?cust qb4o:memberOf gnw:customer ;
      skos:broader ?city .
   ?city gnw:cityGeo ?x .
   BIND (bif:st_area(?x) as ?area) }
```

Line 9. Build the filter statement with the output of the spatial function $\mathtt{f}^{\mathcal{S}}(x)$, construct GP (includes Q') for the outer query, and go to the result (Line 12). At this stage GP is constructed in Lines 2 and 8. The following for the filter statement is added to the GP:

```
FILTER (?cityGeo = ?x) }
```

Line 11. If a spatial literal $v_s \in \mathcal{L}$ is given as the parameter instead, build a filter statement that checks if v_s is equal to the spatial attribute a_s values, and go to the result (Line 12). For example, the following filter condition might be added to graph pattern GP:

```
FILTER (?cityGeo = "POLYGON((10.43951
55.47006, 10.439472 55.470036,
10.439240 (...))")}
```

Line 12. Finally, the algorithm generates query Q, which can be executed over the fact members FM'. In our running examples we obtain the following cases for the generated s-slice query Q.

S-Slice operator with a given spatial value as point data type: The following listing corresponds to the SPARQL output of the running example where the spatial value is given as `POINT` data type (Ex. 12) and filters the level attributes with a spatial predicate within a given level. The graph pattern GP for the query is created in Lines 2 to 7.

```
1 Q = SELECT ?obs WHERE
2     {?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city .
6     ?city gnw:cityGeo ?cityGeo .
7 FILTER (bif:st_within("POINT(10.43951
  55.47006)", ?cityGeo)) }
```

S-Slice operator with a spatial function call: The following listing corresponds to the SPARQL output of the running example where the spatial value is returned from a function call (Ex. 11.2). The graph pattern GP' for the spatial function call is created in Lines 8 to 13. The graph pattern GP for the whole query is created in Lines 2 to 12.

```
1 Q = SELECT ?obs WHERE
2     { ?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city .
6     ?city gnw:cityGeo ?cityGeo .
  # When there is spatial function call we apply
  # inner select for finding the largest city
7     {SELECT ?x (MAX(?area) as ?maxArea) WHERE
8        { ?obs rdf:type qb:Observation ;
```

```
9           gnw:customerID ?cust .
10        ?cust qb4o:memberOf gnw:customer ;
11           skos:broader ?city .
12        ?city gnw:cityGeo ?x .
13     BIND(bif:st_area(?x) as ?area)}
   # Then we apply the filter on the output
   # coming from the spatial function
14 FILTER (?cityGeo = ?x) }
```

S-Slice operator with a given spatial value as polygon data type: The following listing corresponds to the SPARQL output of the running example where the spatial value is given as a `POLYGON` data type (Ex. 11.1) corresponding to a level attribute. The graph pattern *GP* for the query is created in Lines 2 to 7.

```
1 Q = SELECT ?obs WHERE
2     {?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city .
6     ?city gnw:cityGeo ?cityGeo .
7 FILTER (?cityGeo = "POLYGON((10.43951
  55.47006, 10.43947 55.47003, 10.43924 (...))")}
```

S-Dice generator. To generate a SPARQL query for the s-dice operator, $\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}]$ (Def. 18 – parameter $\phi^{\mathcal{S}}$ represents a spatial predicate), we follow the steps sketched in Algorithm 3. The algorithm takes parameter $\phi^{\mathcal{S}}$ as input, which corresponds to a spatial predicate that could represent a topological relation from the $\mathcal{T}_{rel}$ set or a combination of a spatial function (a numeric operation from the $\mathcal{N}_{op}$ set) and a regular predicate ϕ. For illustration, we use the example query that we have introduced in Sect. 5 for s-dice (Ex. 13):

"sales to customers, which are located 5 km distance from their city center". In the following, we discuss the main steps of Algorithm 3 with the running example, where the steps are referencing the line numbers in Algorithm 3.

Line 3. The algorithm runs through the levels from base level l_b to the target spatial level l_s, which are both given in the spatial Boolean predicate $\phi^{\mathcal{S}}$

Line 4. Build the roll-up path for those levels using the helper function `RUPath`. Note that, when we apply the roll-up path to the target level, we can also link the level attributes for the target (spatial) level – as, for example, in the last line of the following listing. The output of function `RUPath` is added to graph pattern *GP*:

```
{?obs rdf:type qb:Observation ;
    gnw:customerID ?cust .
?cust qb4o:memberOf gnw:customer ;
    skos:broader ?city ;
    gnw:customerGeo ?custGeo .
?city gnw:cityGeo ?cityCentGeo .
```

Line 5. Check if $\phi^{\mathcal{S}}$ is to be implemented as a spatial predicate from topological relations $\mathcal{T}_{rel}$ as interpreted in Ex. 13.1.

Line 6. Create a filter statement with a spatial predicate and the spatial level attribute a_s, which is referenced in the roll-up path (Line 4). For our running example, the filter statement is applied on customers that are located within a buffer area of 5 km from their city centers. The spatial predicate `st_within` is used from the topological relations. The following lines are added to graph pattern GP:

```
FILTER (bif:st_within(?custGeo,
?cityCentGeo, 5))}
```

Line 7. Check if $\phi^{\mathcal{S}}$ is to be implemented as a combination of a spatial function $\mathtt{f}^{\mathcal{S}}(x)$ and a regular predicate ϕ as interpreted in Ex. 13.2.

Algorithm 3: *S–DiceGenerator* $(\mathcal{G}^{l}_{(\mathcal{C})}, \phi^{\mathcal{S}}) : Q$

Input: $\mathcal{G}^{l}_{(\mathcal{C})}$, $\phi^{\mathcal{S}}$
Output: Q
1 **begin**
2 $Q = \emptyset$; $GP = \emptyset$
3 **for** $l_b, l_s \in \phi^{\mathcal{S}}$ **do**
4 $GP = GP \cup$ `RUPath`$(\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?\mathrm{a}_s, ?\mathrm{f})$
5 **if** $\phi^{\mathcal{S}}$ *is a spatial predicate* **then**
6 $GP = GP \cup$ (`FILTER` $\phi^{\mathcal{S}}(?\mathrm{a}_s)$)
7 **else if** $\phi^{\mathcal{S}}$ *uses a spatial function* $\mathtt{f}^{\mathcal{S}}(x)$ *and a regular Boolean predicate* ϕ **then**
8 $GP = GP \cup$ (`BIND` $\mathtt{f}^{\mathcal{S}}(x)$`AS` $?\mathrm{v}_x$) $GP = GP \cup$ (`FILTER` $\phi(?\mathrm{v}_x)$)
10 **return** Q = `SELECT` ?f `WHERE` GP

Lines 8, 9. Create a bind statement based on a spatial function (i.e., calculate `st_distance` between customers and city center) and a filter statement based on the assigned values with a regular predicate (i.e., less than 5 km). The following lines are added to graph pattern GP:

```
BIND (bif:st_distance (?custGeo, ?cityCentGeo)
   AS ?distance) FILTER ( ?distance < 5 )}
```

Line 10. Generate query Q for selecting the facts $f \in FM'$ matching the incrementally created graph pattern GP in the previous steps. In our running examples we obtain the following cases for the generated s-dice query Q.

S-Dice operator with $\phi^{\mathcal{S}}$: The following listing is the SPARQL query generated for the running example (Ex. 13.1), where the spatial predicate is interpreted as a topological relation. The graph pattern GP for the query is created in Lines 2 to 8.

```
1 Q = SELECT ?obs WHERE
2     { ?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city ;
6        gnw:customerGeo ?custGeo .
7     ?city gnw:cityGeo ?cityCentGeo .
8 FILTER (bif:st_within (?cityCentGeo, ?custGeo, 5))}
```

S-Dice operator with $\mathtt{f}^{\mathcal{S}}(x)$ and a regular Boolean predicate ϕ: The following listing is the SPARQL query generated for the running example (Ex. 13.2), where the spatial predicate is interpreted as a combination of a spatial function and a regular predicate. The graph pattern GP for the query is created in Lines 2 to 9.

```
1 Q = SELECT ?obs WHERE
2     { ?obs rdf:type qb:Observation ;
3        gnw:customerID ?cust .
4     ?cust qb4o:memberOf gnw:customer ;
5        skos:broader ?city ;
6        gnw:customerGeo ?custGeo .
7     ?city gnw:cityGeo ?cityCentGeo .
8 BIND (bif:st_distance (?custGeo, ?cityCentGeo) AS
9     ?distance) FILTER ( ?distance < 5 ) }
```

S-Roll-up Generator. To generate a SPARQL query for the s-roll-up operator from a high-level SOLAP expression, $\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)]$ (Def. 19), where parameter $\mathtt{f}^{\mathcal{S}}(L(d_i))$ denotes a spatial function on spatial level members and $agg(m)$ is an aggregate function on measures. For illustration, we use the query example for s-roll-up given in Sect. 5 for s-roll-up (Ex. 14): "total amount of sales to customers by city of the closest suppliers". We follow the main steps sketched in Algorithm 4 in the following.

Lines 2, 3. Build the roll-up path using helper function `RUPath`. In addition to the variables given in the `RUPath` function, we also need to consider measures and measure value variables (Line 3) since we aggregate the measures. A measure is specified in the following listing of the running example as `gnw:salesAmount`. The following lines are added to the graph pattern GP:

```
{?obs rdf:type qb:Observation ;
    gnw:customerID ?cust ;
    gnw:supplierID ?sup ;
    gnw:salesAmount ?sales .
?cust qb4o:memberOf gnw:customer ;
    gnw:customerGeo ?custGeo ;
    skos:broader ?city .
?sup qb4o:memberOf gnw:supplier ;
    gnw:supplierGeo ?supGeo ;
    skos:broader ?city .
?city qb4o:memberOf gnw:city .
```

Line 4. Build inner select subquery to apply the spatial function $\mathtt{f}^{\mathcal{S}}$ on the spatial level members $L(d_i)$ (i.e., Customer, Supplier). In the example, we will use this information to create a dynamic spatial hierarchy from the Customer to the City level.

Algorithm 4: *SRUGenerator*$(\mathcal{G}^{I}_{(\mathcal{C})}, \mathtt{f}^{\mathcal{S}}(L(d_s)), agg(m)) : Q$

Input: $\mathcal{G}^{I}_{(\mathcal{C})}, \mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)$
Output: Q
1 **begin**
2 $\quad Q = \emptyset$; GP = `RUPath`$(\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?\mathrm{a}_s, ?\mathrm{f})$
3 $\quad GP = GP \cup$ (?f $id^{\mathcal{S}}(m)$?m)
4 $\quad$ **for** $\mathtt{f}^{\mathcal{S}}(L(d_s))$ **do**
5 $\quad\quad GP' =$ `RUPath`$(\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?\mathrm{a}_s, ?\mathrm{f})$;
$\quad\quad Q' = \emptyset$
6 $\quad\quad GP' = GP' \cup$ (`BIND` $\mathtt{f}^{\mathcal{S}}(x)$ `AS` $?\mathrm{v}_x$)
7 $\quad\quad Q'$ = `SELECT` ?x (`AGG`($?\mathrm{v}_x$) `AS` $?\mathrm{v}_y$) `WHERE` GP' `GROUP BY` ?x
8 $\quad\quad GP = GP \cup Q' \cup$ (`FILTER` ?x = $?\mathrm{a}_s$ && $\mathtt{f}^{\mathcal{S}}(x)$ = $?\mathrm{v}_y$)
9 $\quad\quad$ let $l_s = l'_s$
10 **return** Q = `SELECT` ?f $?\mathrm{l}'_s$ `AGG`(?m) `WHERE` GP `GROUP BY` ?f $?\mathrm{l}'_s$

Line 5. Call `RUPath` for the inner select subquery to link the geometry attributes of base level members with different variables and create a

graph pattern GP' for the inner select. The following lines are added to the graph pattern GP':

```
{?obs rdf:type qb:Observation ;
    gnw:customerID ?cust1 ;
    gnw:supplierID ?sup1 .
?sup1 gnw:supplierGeo ?sup1Geo .
?cust1 gnw:customerGeo ?cust1Geo .
```

Line 6. Build the bind statement in order to calculate the spatial function $\mathtt{f}^{\mathcal{S}}(L(d_s))$ on spatial level members. For the running example the spatial function is `st_distance`. The following lines are added to the graph pattern GP':

```
BIND (bif:st_distance(?cust1Geo, ?sup1Geo)
AS ?distance)}
```

Line 7. Generate the inner select query Q' using graph pattern GP' (Lines 5 and 6). Select the corresponding level members (Customer level for the running example) and group them in a group by statement on the selected level members. Note that this is where the spatial function $\mathtt{f}^{\mathcal{S}}(L(d_i))$ is called with a wrapper expression (e.g., MIN, MAX, etc.) to find the closest distance. The following lines illustrate the inner select query Q':

```
Q‘ = {SELECT ?cust1 (MIN(?distance) AS
?minDistance) WHERE GP‘
   GROUP BY ?cust1}
```

Lines 8, 9. Build the filter statement for the whole query based on the output of the spatial function, which is calculated in the inner select subquery. Then, add the filter and inner select subquery to the main graph pattern GP' (Line 8). The filter statement for the running example is :

```
FILTER (?cust = ?cust1 && bif:st_distance
(?custGeo, ?supGeo) = ?minDistance)}
```

Note that in Line 9, the spatial target level l_s (City) is altered to a dynamic spatial level l'_s since applying the spatial function creates a dynamic hierarchy.

Line 10. Generate query Q for computing the facts $f \in FM'$ based on graph pattern GP created in the previous steps. The measures are also aggregated at the spatial target level (closest City, which is dynamically

selected). The group by statement is applied on the fact members and target level members. In our running example we obtain the following case for the generated s-roll-up query Q.

S-Roll-up operator: The following listing shows the generated SPARQL query. Graph pattern GP' for the inner select subquery is created in Lines 15 to 22 and the graph pattern GP for the whole query is created in Lines 3 to 24.

```
1  Q = SELECT ?obs ?city (SUM(?sales) AS
2  ?totalSales) WHERE
3  { ?obs rdf:type qb:Observation ;
4      gnw:customerID ?cust ;
5      gnw:supplierID ?sup ;
6      gnw:salesAmount ?sales .
7  ?cust qb4o:memberOf gnw:customer ;
8      gnw:customerGeo ?custGeo ;
9      gnw:customerName ?custName ;
10     skos:broader ?city .
11 ?city qb4o:memberOf gnw:city .
12 ?sup gnw:supplierGeo ?supGeo .
   # Inner Select for the distance function
13     { SELECT ?cust1 (MIN(?distance) AS
14     ?minDistance) WHERE
15     { ?obs rdf:type qb:Observation ;
16        gnw:customerID ?cust1 ;
17        gnw:supplierID ?sup1 .
18     ?sup1 gnw:supplierGeo ?sup1Geo .
19     ?cust1 gnw:customerGeo ?cust1Geo .
20     BIND (bif:st_distance( ?cust1Geo, ?sup1Geo )
21     AS ?distance)}
22     GROUP BY ?cust1 }
23     FILTER (?cust = ?cust1 && bif:st_distance
24     (?custGeo, ?supGeo) = ?minDistance)}
25 GROUP BY ?city ?obs
```

S-Drill-down Generator. The semantics of the s-drill-down operator are defined in the same way as for the s-roll-up operator with the condition that the input cube $\mathcal{C}$ for s-roll-up is obtained using a function *BaseCube* such that $SDD(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)] = SRU(BaseCube(\mathcal{C}))[\mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m)]$ (Def. 20). Therefore, no generator algorithm and steps are specified since an s-drill-down operator corresponds to a rewriting of an s-roll-up operator, which is obtained with a *Base* function that calls the base cube graph in `SRUGenerator` as; `SDDGenerator` $=$ `SRUGenerator` $(Base(\mathcal{G}^I_{(\mathcal{C})}), \mathtt{f}^{\mathcal{S}}(L(d_i)), agg(m))$.

6.2 Nested SOLAP operations to SPARQL

We now show how a SPARQL query can be generated for a nested SOLAP expression. In general, a nested set of SOLAP operators can be rewritten into an expression with an additional s-dice, on top of a series of s-roll-ups, on top of one or more s-slices, on top of an s-dice, i.e., *(s-dice$_2$(s-roll-up$_1$(... s-roll-up$_k$(s-slice$_1$(... s-slice$_n$(s-dice$_1$(C)))))))*.

Let us begin with a simpler nested form that shows the most typical pattern, namely *(s-roll-up (s-slice (s-dice(C))))*, where initially a subcube graph is selected by s-dice. Afterwards, an s-slice is performed on a higher level of a dimension. Then, an s-roll-up is applied, which aggregates the measures in the sliced cube from a lower level to a higher level. Finally, we could also perform another s-dice for filtering the measures. There may be several s-slices and s-roll-ups in between.

We formulate the nested SOLAP query as ^{3}*(s-roll-up* ^{2}*(s-slice* ^{1}*(s-dice(C))))* and apply our running examples such that the enumeration of operators can be interpreted as follows: ^{1}Get the subcube graph of customers that are located within a 5 km distance from their city center, ^{2}slice on the customers of the largest country, (which drops the dimension and leaves out all the other countries) and ^{3}get the total amount of sales for customers by the city of their closest suppliers (aggregates the measure Sales amount from Customer to Closest City level). Finally, we may also perform another (s-)dice on measures, e.g., filtering the total amount of sales greater than 10500. To perform nested SOLAP operators, we identify a set of principles to be considered by the algorithm.

Principle 1: Perform s-dice in the beginning or at the end.

Principle 2: If there are several s-roll-up or s-slice operations call their generator algorithms repeatedly.

Principle 3: Always separate `FILTER` clauses when a SOLAP generator algorithm is used. Enumerate separated `FILTER` clauses. If a SOLAP operator is the final function added to the graph pattern, do not separate the `FILTER` clause.

Principle 4: Build the final graph pattern with the separated and enumerated `FILTER` clauses with respect to Principle 3.

Principle 5: Drop the main `SELECT` clause from each SOLAP generator algorithms and build only one `SELECT` that is added to the query at the end.

Principle 6: Separate the `GROUP BY` clause and `AGG` functions from the s-roll-up generator algorithms (and enumerate them), and build add them to the main (outer) `SELECT` clause at the end.

Algorithm 5: *WriteSPARQL*$((\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_i)),\mathrm{agg(m)}](\mathcal{SS}(\mathcal{C})[l_b, l_s, v_{a_s}](\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}])))):Q$

Input: $(\mathcal{SRU}(\mathcal{C})[\mathtt{f}^{\mathcal{S}}(L(d_s)), agg(m)](\mathcal{SS}(\mathcal{C})[l_b, l_s, v_{a_s}](\mathcal{SD}(\mathcal{C})[\phi^{\mathcal{S}}])))$
Output: Q

1 **begin**
2 $Q = \emptyset$; GP = `RUPath`$(\mathcal{G}^{S}_{(\mathcal{C})}, l_b, l_s, a_{ID}, ?\mathrm{a}_s, ?\mathrm{f})$
3 $GP = GP \cup (?\mathrm{f}\ id^{S}(m)\ ?\mathrm{m}\ )$
4 GP^1 = `S-DiceGenerator`$(\mathcal{G}^{I}_{(\mathcal{C})}, \phi^{\mathcal{S}})$ \ `FILTER`^{1} \ `SELECT`
5 $GP = GP \cup GP^1$
6 GP^2 = `S-SliceGenerator`$(\mathcal{G}^{I}_{(\mathcal{C})}, v_s, l_b, l_s)$ \ `FILTER`^{2} \ `SELECT`
7 $GP = GP \cup GP^2$
8 $GP = GP \cup$ `FILTER`$^1 \cup$ `FILTER`$^2 \cup$
9 `SRUGenerator`$(\mathcal{G}^{I}_{(\mathcal{C})}, \mathtt{f}^{\mathcal{S}}(L(d_s)), agg(m))$ \ `SELECT` \ `GROUP BY`^{1} \ `AGG`^{1}
10 **return** Q = `SELECT` $?\mathrm{l}'_s$ `AGG`$^1(?\mathrm{m})$ `WHERE` GP `GROUP BY`^{1} $?\mathrm{l}'_s$

To separate the `FILTER` clauses, we call SOLAP generator algorithms without their `FILTER` clause and enumerate each `FILTER` clause for each SOLAP generator algorithm that is used, i.e., `S-SliceGenerator` $(\mathcal{G}^{I}_{(\mathcal{C})}, \phi^{\mathcal{S}})$ \ `FILTER`^{1} (Algorithm 5, Line 4). Then, we build the final graph pattern with these separated `FILTER` clauses i.e., $GP = GP \cup$ `FILTER`$^1 \cup$ `FILTER`^{2} (Line 8). When the last SOLAP generator algorithm is called, the output is directly added to the graph pattern without separating its `FILTER` clause (Line 9). Throughout the algorithm, all the `SELECT` clauses are omitted and combined into one `SELECT` in the output on Line 10. According to Principle 6, if there are any `GROUP BY` clauses and `AGG` functions (on measures) in inner selects, we eliminate them with " \ " from the inner selects (Line 9) and finally build the main (outer) select query with (Line 10). Note that in the algorithm, the general graph pattern GP is initially created by the `RUPath` function (Line 2) and incremented with triple patterns for selected measures (Line 3).

Example 16. ((^{3}*s-roll-up* (^{2}*s-slice* (^{1}*s-dice*($\mathcal{C}$))))):
^{1}Get the subcube graph of customer that are located within a 5 km distance from their city center, ^{2}slice on the customers of the largest country, (which drops the dimension and leave out all the other countries) and ^{3}get the total amount of sales of customers by the city of their closest suppliers (aggregates the measure Sales amount from Customer to Closest City level). The query is written starting from the innermost operator s-dice to the outermost operator s-roll-up.

```
SELECT ?city (SUM(?sales) AS ?totalSales)
WHERE {
    ?obs rdf:type qb:Observation ;
       gnw:customerID ?cust ;
       gnw:supplierID ?sup ;
       gnw:salesAmount ?sales .
    ?cust qb4o:memberOf gnw:customer ;
       gnw:customerGeo ?custGeo ;
       skos:broader ?city .
    ?sup qb4o:memberOf gnw:supplier ;
       gnw:supplierGeo ?supGeo ;
       skos:broader ?city .
    ?city qb4o:memberOf gnw:city ;
       gnw:cityGeo ?cityCentGeo ;
       skos:broader ?country .
    ?country qb4o:memberOf gnw:country ;
       gnw:countryGeo ?countGeo .
    ?city gnw:cityGeo ?cityGeo .
### 1.Inner select for (S-SLICE)
### Find the largest country
    {SELECT ?x (MAX(?area) as ?maxArea)
    WHERE {
       ?obs rdf:type qb:Observation ;
          gnw:customerID ?cust .
       ?cust qb4o:memberOf gnw:customer ;
          skos:broader ?city .
       ?city skos:broader ?country .
       ?country gnw:countryGeo ?x .
    BIND(bif:st_area(?x) as ?area)}}
### 2.Inner select for (S-ROLL-UP)
### Find the closest suppliers to customers
    { SELECT ?cust1 (MIN(?distance) AS
    ?minDistance) WHERE {
       ?obs rdf:type qb:Observation ;
          gnw:customerID ?cust1 ;
          gnw:supplierID ?sup1 .
       ?sup1 gnw:supplierGeo ?sup1Geo .
       ?cust1 gnw:customerGeo ?cust1Geo .
    BIND (bif:st_distance(?cust1Geo, ?sup1Geo)
    AS ?distance)}
    GROUP BY ?cust1 }
### FILTER for S-DICE, to get a subcube
FILTER (bif:st_within (?custGeo, ?cityCentGeo, 5))
### FILTER for S-SLICE, the 1st inner SELECT
FILTER (?countGeo = ?x)
### FILTER for S-ROLL-UP, the 2nd inner SELECT
```

```
39 FILTER (?cust = ?cust1 && bif:st_distance
   (?custGeo, ?supGeo) = ?minDistance)}
40 GROUP BY ?city
```

The graph pattern *GP* is initially created with the `RUPath` function for the corresponding levels and level attributes (Lines 3 to 18 of the generated SPARQL query in the listing above). The first operator is called by function `S-DiceGenerator`, where the first `FILTER` clause of the outer selected is added to the query (Line 37). The second operator is called by the `S-SliceGenerator` function excluding its `FILTER` clause (Lines 19 to 27), which is followed by the `SRUGenerator` function without `GROUP BY` and `AGG` statements (Lines 28 to 36). Note that in Line 36, `GROUP BY` is applied on the lower level Customer, and the actual `GROUP BY` for the target City level is applied in the last line (Line 40). Separated `FILTER` clauses for the `S-DiceGenerator` and `S-SliceGenerator` functions are later added to the graph pattern (Algorithm 5, Line 8) corresponding to Lines 37 and 38 in the above example. The main outer select query is defined in the first line by specifying the target level (City) and aggregate function on measures (sum of the total Sales).

7 Conclusions and future work

Motivated by the need for a formal foundation for spatial data warehouses on the Semantic Web, this paper made a number of contributions. First, it proposed the QB4SOLAP vocabulary (metamodel), which supports spatially enhanced multidimensional (MD) data cubes over RDF data. This allows users to publish MD spatial data in RDF format. Second, the paper defines a number of spatial OLAP (SOLAP) operators over the defined QB4SOLAP cubes, allowing spatial analytical queries over RDF data, and gives their formal semantics. Third, the paper provides algorithms for generating spatially extended SPARQL queries from individual and nested SOLAP operators, allowing users to write their spatial analytical queries in our high-level SOLAP language instead of the lower-level and more complex SPARQL. Fourth, the vocabulary, operators, and query generation algorithms are validated by applying them to a realistic use case.

Fig. C.8 presents our future vision of SOLAP on the Semantic Web with regards to the current state of the art and ongoing work. In order to verify the algorithms, which are defined in this paper, we developed *GeoSemOLAP* [32], a SPARQL query generation tool. GeoSemOLAP allows users to perform SOLAP operations and generate SPARQL queries interactively through a GUI. We refer to our screencast[16] for a detailed demonstration of GeoSemOLAP.

[16] `https://youtu.be/Pc3RBPPgBhA`

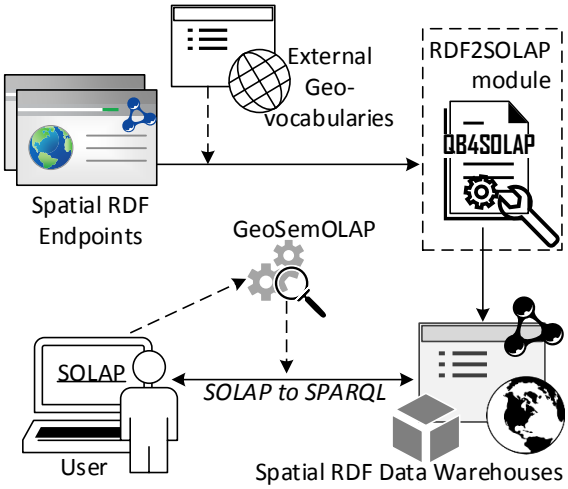


Fig. C.8: Our future vision of SOLAP on the SW

Publishing spatial DWs on the SW allows users to also exploit the existing external geo-vocabularies (e.g., GeoNames, etc.)[17] by defining spatial levels and hierarchies from external open data sources. Our main ongoing work thus focuses on developing an RDF2SOLAP enrichment module that performs the multidimensional annotation of existing spatial RDF datasets with QB4SOLAP in a (semi-)automatic fashion.

Additional interesting aspects of future work would be, for instance, extending the formal techniques and algorithms for generating SOLAP queries in SPARQL to work over multiple RDF cubes, i.e., to support *s-drill-across*, and supporting spatial aggregation (*s-aggregation*) with user-defined functions over spatial measures. It would be also interesting to increase efficiency by extending our spatial data warehouse with techniques that been developed in the context of RDF data cubes and SPARQL analytical queries in general, e.g., materialization and optimizing the physical layout [33–35], and to enable efficient support of a broad range of external sources by considering aspects such as federated processing of analytical queries [36] and schema heterogeneity [37]. We will also consider more efficient representations of the data, e.g., by removing redundancies. Furthermore, it would be interesting to extend QB4SOLAP and GeoSemOLAP [32] to handle highly dynamic spatio-temporal data and queries, as for instance, found in large-scale transport

[17] GeoNames: `http://www.geonames.org/`
Global Administrative Areas: `http://gadm.geovocab.org/`
NUTS – EU's Nomenclature of Territorial Units for Statistics: `http://nuts.geovocab.org/`

analytics [38].

A Appendix

A.1 Query Run Times

Table C.5 presents the query runtimes for the SOLAP operator examples (Ex. 12, Ex. 13.1, Ex. 13.2, Ex. 14, and Ex. 15) given in Sect. 5 and a nested SOLAP operator example (Ex. 16) given in Sect. 6.2. The SOLAP queries are tested against an instance of the use case dataset (GeoNorthwind) that we have discussed in Sect. 4. In total, *48677* triples are obtained from the GeoNorthwind dataset. The published RDF graph instance is denoted as $\mathcal{C}$ in Table C.5. We used Virtuoso Open Source Edition (Column Store and multi threaded) Version 7.2.5 on an Ubuntu 14.04 server with 2.30GHz CPU and 16 GB RAM to publish the triples at the SPARQL endpoint `http://lod.cs.aau.dk:8890/sparql`. The actual queries used in the paper are available at `http://extbi.cs.aau.dk/SOLAP4SW/queries`.

Table C.5: Runtimes in seconds

SOLAP Operators	Query Runtime
Ex. 12 *(s-slice($\mathcal{C}$))*	0.07
Ex. 13.1 *(s-dice($\mathcal{C}$))*	0.09
Ex. 13.2 *(s-dice($\mathcal{C}$))*	1.01
Ex. 14 *(s-roll-up($\mathcal{C}$))*	2.03
Ex. 15 *(s-drill-down($\mathcal{C}$))*	1.86
Ex. 16 *(s-roll-up (s-slice (s-dice($\mathcal{C}$))))*	3.04

The query runtime of the s-slice query in Ex 12 is measured as 0.07 seconds, which is an efficient SOLAP operator to execute, since filtering the given spatial literal is done only for the records of the specified level instances.

The query runtimes of the s-dice queries in Ex. 13.1 and Ex. 13.2 are measured as 0.09 and 1.01 seconds respectively, for the two alternative ways of implementing s-dice in SPARQL. It is clear that the first use of the s-dice operator is more efficient compared to the second, even though both return the same results. The first one performs s-dice by filtering the instances of the customers through their geometries that are within a buffer area of a circle with 5 km radius and the city center geometry as the center point of the circle. The second one performs s-dice by measuring the distances between each customer's location and their city center, and then applies a filter of those measurements to find the ones that are less than 5 km. This is a more

expensive approach due to measuring the distances between each customer and city instances with a spatial distance function.

The query runtime of the s-roll-up query in Ex. 14 is measured as 2.03 seconds. The s-roll-up operator has a longer response time than the other operators as it combines a spatial distance function with aggregate operators. Due to its complexity, s-roll-up requires an inner select where it binds the distances of specified spatial level attributes (e.g., customer and supplier geometry), which is calculated with a spatial distance function. Those distance measurements are then filtered in the outer select with respect to the aggregate function.

The query runtime of the s-drill-down query in Ex. 15 is measured as 1.86 seconds. S-drill-down operates in a very similar manner as s-roll-up. Theoretically, s-drill-down operates on an already spatially rolled up data cube, whilst it is implemented in practice as an s-roll-up on the base cube. The RDF data is at the lowest granularity, which is equal to the definition of the base cube. The SPARQL query of the s-drill-down operator is very similar the s-roll-up operator. The difference of the query runtime between Ex. 14 and Ex. 15 is directly related to the number of instances of the specified levels and the complexity of the spatial functions.

The query runtime for the nested SOLAP query in Ex. 16 is measured as 3.04 seconds. The nesting in the query is the main reason why it has a longer response time. Moreover, the nested query has more triple patterns and clauses that need to be evaluated during query execution. Hence, it takes longer than evaluating a single SOLAP operator.

A.2 Table of Contents

Table C.6 summarizes a list of all the definitions, extensions, remarks, and algorithms. *Definitions* are enumerated starting from "1". *Extensions* are enumerated starting from "5" to be consistent with the base definition of the extension, e.g., Def. 5 (Dimensions) is followed by Ext. 5 (Spatial dimensions). *Remarks* are enumerated starting from "17" for OLAP operators, which are given before the definitions of the corresponding SOLAP operator, e.g., Remark 17 (Slice) is followed by Def. 17 (S-Slice). *Algorithms* are enumerated starting from "1".

References

[1] N. Gür, K. Hose, E. Zimányi, and T. B. Pedersen, “Modeling and Querying Spatial Data Warehouses on the Semantic Web,” in *Semantic Technology: 5th Joint International Semantic Technology Conference (JIST'15)*,

vol. 9544. Springer, 2015, pp. 1–20, https://dx.doi.org/10.1007/978-3-319-31676-5_1.

[2] N. Gür, K. Hose, T. B. Pedersen, and E. Zimányi, "Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP," in *Semantic Technology: 6th Joint International Semantic Technology Conference (JIST'16)*, vol. 10055. Springer, 2016, pp. 287–304, https://dx.doi.org/10.1007/978-3-319-50112-3_22.

[3] A. Abelló, O. Romero, T. Pedersen, R. Berlanga Llavori, V. Nebot, M. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, no. 2, pp. 571–588, 2014, https://doi.org/10.1109/TKDE.2014.2330822.

[4] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *The Semantic Web: ESWC 2012 Satellite Events*, vol. 7540. Springer, 2012, pp. 87–101, https://dx.doi.org/10.1007/978-3-662-46641-4_7.

[5] R. Cyganiak, D. Reynolds, and J. Tennison, "The RDF Data Cube Vocabulary," 2014.

[6] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web using QB4OLAP," in *Data Warehousing and Knowledge Discovery (DaWaK'14)*, vol. 8646. Springer, 2014, pp. 45–56, https://dx.doi.org/10.1007/978-3-319-10160-6_5.

[7] R. P. Deb Nath, K. Hose, and T. B. Pedersen, "Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses," in *Proceedings of the 18th International Workshop on Data Warehousing and OLAP (DOLAP'15)*. ACM, 2015, pp. 15–24, https://doi.org/10.1145/2811222.2811229.

[8] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen, "Dimensional enrichment of statistical linked open data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 40, pp. 22–51, 2016, https://dx.doi.org/10.1016/j.websem.2016.07.003.

[9] P. Revesz, *Introduction to Databases: From Biological to Spatio-Temporal*. Springer, 2009.

[10] Y. Bédard, E. Bernier, S. Larrivée, M. Nadeau, M. Proulx, and S. Rivest, "Spatial OLAP," in *Forum annuel sur la RD, Géomatique VI: Un monde accessible*, 1997, pp. 13–14.

[11] S. Rivest, Y. Bédard, and P. Marchand, "Toward better support for spatial decision making: defining the characteristics of spatial on-line analytical processing (SOLAP)," *GEOMATICA*, vol. 55, no. 4, pp. 539–555, 2001, canadian Institute of Geomatics.

[12] E. Edoh-Alove, S. Bimonte, and F. Pinet, "An UML Profile and SOLAP Datacubes Multidimensional Schemas Transformation Process for Datacubes Risk-Aware Design," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 11, no. 4, pp. 64–83, 2015, https://dx.doi.org/10.4018/ijdwm.2015100104.

[13] T. B. Pedersen and N. Tryfona, "Pre-aggregation in Spatial Data Warehouses," in *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD'01)*. Springer, 2001, pp. 460–478, http://dx.doi.org/10.1007/3-540-47724-1_24.

[14] I. V. Lopez, R. T. Snodgrass, and B. Moon, "Spatiotemporal aggregate computation: A survey," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 17, no. 2, pp. 271–286, 2005, https://doi.org/10.1109/TKDE.2005.34.

[15] J. da Silva, V. C. Times, A. C. Salgado, C. Souza, R. d. N. Fidalgo, and A. G. de Oliveira, "A set of aggregation functions for spatial measures," in *Proceedings of the 11th International Workshop on Data Warehousing and OLAP (DOLAP'08)*. ACM, 2008, pp. 25–32, https://doi.acm.org/10.1145/1458432.1458438.

[16] L. I. Gómez, S. Haesevoets, B. Kuijpers, and A. A. Vaisman, "Spatial aggregation: Data model and implementation," *Information Systems (IS)*, vol. 34, no. 6, pp. 551–576, 2009, https://dx.doi.org/10.1016/j.is.2009.03.002.

[17] J. Han, N. Stefanovic, and K. Koperski, "Selective Materialization: An Efficient Method for Spatial Data Cube Construction," in *Research and Development in Knowledge Discovery and Data Mining (PAKDD'98)*. Springer, 1998, pp. 144–158, https://dx.doi.org/10.1007/3-540-64383-4_13.

[18] E. Malinowski and E. Zimányi, *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Data-Centric Systems and Applications*. Springer, 2008, https://dx.doi.org/10.1007/978-3-540-74405-4.

[19] A. Vaisman and E. Zimányi, "A Multidimensional Model Representing Continuous Fields in Spatial Data Warehouses," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'09)*. ACM, 2009, pp. 168–177, https://doi.acm.org/10.1145/1653771.1653797.

[20] L. I. Gómez, S. A. Gómez, and A. A. Vaisman, "A Generic Data Model and Query Language for Spatiotemporal OLAP Cube Analysis," in *Proceedings of the 15th International Conference on Extending Database Technology (EDBT'12)*. ACM, 2012, pp. 300–311, https://doi.acm.org/10.1145/2247596.2247632.

[21] R. Battle and D. Kolas, "Enabling the Geospatial Semantic Web with Parliament and GeoSPARQL," *Semantic Web Journal (SWJ)*, vol. 3, no. 4, pp. 355–370, 2012, https://dx.doi.org/10.3233/SW-2012-0065.

[22] K. Kyzirakos, M. Karpathiotakis, and M. Koubarakis, "Strabon: A Semantic Geospatial DBMS," in *The Semantic Web: 11th International Semantic Web Conference (ISWC'12)*. Springer, 2012, pp. 295–311, https://dx.doi.org/10.1007/978-3-642-35176-1_19.

[23] M. Koubarakis, M. Karpathiotakis, K. Kyzirakos, C. Nikolaou, and M. Sioutis, "Data Models and Query Languages for Linked Geospatial Data," in *Reasoning Web. Semantic Technologies for Advanced Query Answering*. Springer, 2012, pp. 290–328, https://dx.doi.org/10.1007/978-3-642-33158-9_8.

[24] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "LinkedGeoData: A Core for a Web of Spatial Open Data," *Semantic Web Journal (SWJ)*, vol. 3, pp. 333–354, 2012, https://dx.doi.org/10.3233/SW-2011-0052.

[25] G. Rojas, G. Giannopoulos, and J. J. L. Daniel Hladky, "Managing Geospatial Linked Data in the GeoKnow Project," in *The Semantic Web in Earth and Space Science. Current Status and Future Directions*, vol. 20. IOS Press, 2015, p. 51.

[26] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Semantic Technology: 4th Joint International Semantic Technology Conference (JIST'14)*, vol. 8943. Springer, 2014, pp. 178–186, https://dx.doi.org/10.1007/978-3-319-15615-6_13.

[27] D. A. Randell, Z. Cui, and A. G. Cohn, "A spatial logic based on regions and connection," in *Principles of Knowledge Representation and Reasoning*, vol. 92, 1992, pp. 165–176.

[28] M. J. Egenhofer and J. Herring, "A mathematical framework for the definition of topological relationships," in *Fourth international symposium on spatial data handling*. Zurich, Switzerland, 1990, pp. 803–813.

[29] M. Perry and J. Herring, "GeoSPARQL: A Geographic Query Language for RDF Data," *OGC Implementation Standard*, 2012.

[30] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson, "A Foundation for Capturing and Querying Complex Multidimensional Data," *Information Systems (IS)*, vol. 26, no. 5, pp. 383–423, 2001, http://dx.doi.org/10.1016/S0306-4379(01)00023-0.

[31] C. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi, "Cube algebra: A Generic User-centric Model and Query Language for OLAP Cubes," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 9, no. 2, pp. 39–65, 2013, http://dx.doi.org/10.4018/jdwm.2013040103.

[32] N. Gür, J. Nielsen, K. Hose, and T. B. Pedersen, "GeoSemOLAP: SOLAP on the Semantic Web Made Easy," in *Proceedings of the 26th International Conference Companion on World Wide Web (WWW'17)*. ACM, 2017, https://dx.doi.org/10.1145/3041021.3054731.

[33] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu, "View Selection in Semantic Web Databases," *VLDB Endowment*, vol. 5, no. 2, pp. 97–108, 2011, https://dx.doi.org/10.14778/2078324.2078326.

[34] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi, "Optimizing Aggregate SPARQL Queries Using Materialized RDF Views," in *The Semantic Web: 15th International Semantic Web Conference (ISWC'16)*. Springer, 2016, pp. 341–359, https://dx.doi.org/10.1007/978-3-319-46523-4_21.

[35] K. A. Jakobsen, A. B. Andersen, K. Hose, and T. B. Pedersen, "Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries," in *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD'15)*, 2015, http://ceur-ws.org/Vol-1426/paper-02.pdf.

[36] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi, "Processing Aggregate Queries in a Federation of SPARQL Endpoints," in *The Semantic Web: 12th European Semantic Web Conference (ESWC'15)*. Springer, 2015, pp. 269–285, https://dx.doi.org/10.1007/978-3-319-18818-8_17.

[37] J. Rouces, G. de Melo, and K. Hose, "FrameBase: Representing N-Ary Relations Using Semantic Frames," in *The Semantic Web: 12th European Semantic Web Conference (ESWC'15)*. Springer, 2015, pp. 505–521, https://dx.doi.org/10.1007/978-3-319-18818-8_31.

[38] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler, "Highly scalable trip grouping for large-scale collective transportation systems," in *Proceedings of the 11th International Conference on Extending Database Technology (EDBT'08)*. ACM, 2008, pp. 678–689, https://doi.acm.org/10.1145/1353343.1353425.

Table C.6: Table of Contents

Paper D

GeoSemOLAP: Geospatial OLAP on the Semantic Web Made Easy

Nurefşan Gür, Jacob Nielsen, Katja Hose, and Torben Bach Pedersen

The (demo) paper has been published in the
Proceedings of the 26th International Conference on World Wide Web and
awarded as *Best Demo* by Reviewer's Choice.
ACM ID: 3054731, pp. 213–217, 2017. DOI: 10.1145/3041021.3054731

Abstract

The proliferation of spatial data and the publication of multidimensional (MD) data on the Semantic Web (SW) has led to new opportunities for On-Line Analytical Processing (SOLAP) over spatial data using SPARQL. However, formulating such queries results in verbose statements and can easily become very difficult for inexperienced users. Hence, we have developed GeoSemOLAP to enable users without detailed knowledge of RDF and SPARQL to query the SW with SOLAP. GeoSemOLAP generates SPARQL queries based on high-level SOLAP operators and allows the user to interactively formulate queries using a graphical interface with interactive maps.

1 Introduction

The data that is currently available on the Semantic Web (SW) has evolved from being simple, mostly alphanumeric, to also include more complex data such as spatial data [1]. Although spatial data is common on the SW, it remains difficult to utilize and analyze because spatial data requires special techniques for encoding it in RDF and evaluating spatial functions, which are often not supported by standard triple stores. On the other hand, the support of spatial data is more common in the area of relational databases, where spatial data warehouses are typically based on a multidimensional (MD) relational model involving spatial dimensions. Efficiently processing spatial data in this context is typically realized by Online Analytical Processing (OLAP) extended to support spatial analyses (SOLAP) [2].

However, with the growing popularity of the Linked Open Data (LOD) movement in the public sector, more and more spatial datasets from governmental domains [3] are becoming available on the SW. The availability of such datasets has led to novel opportunities for decision makers to analyze the growing public data with analytical data warehouse queries on SW data [4]. However, these potential users are only in very rare cases sufficiently familiar with the underlying technologies that are necessary to actually perform the desired analyses.

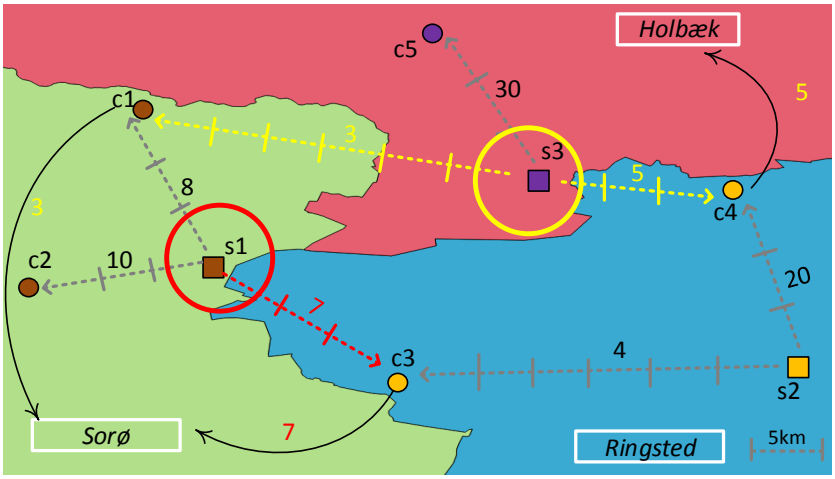


Fig. D.1: Example map of sales data

Let us consider an example scenario: Fig. D.1 shows a map highlighting the amount of sales between customers *(c1, c2, . . . , c5)* and suppliers *(s1, s2, s3)* in three Danish cities *(Sorø, Holbæk, Ringsted)*. Let us assume that

an analyst wants to obtain the total sales of customers grouped by cities of their closest supplier – Query E.1 shows the corresponding example query formulated in SPARQL. This means that, first, for each customer we need to determine the closest supplier and the city in which the supplier is located. Then, we create one group for each of these cities and compute the total sales per city.

For each sales event, the dataset contains information about the involved customer and supplier; and for each customer and supplier the dataset contains the city they are located in. As the dataset does not contain any information about the distances between customers and suppliers, we have to use a *spatial function* (distance in this example) and evaluate it during runtime.

Based on the obtained information, we can aggregate the results as described above. Technically, this corresponds to a SOLAP operator: *s-roll-up* [2, 5].

As we can see in Query E.1, formulating a roll-up to a higher level, e.g., from sales by supplier to sales by city, in a SPARQL query involves several triple patterns. Extending such a query with spatial aspects (as necessary for s-roll-up) requires even more triple patterns and spatial functions (such as distance), which can easily become overwhelming for inexperienced users.

```
SELECT ?obs ?supCity (SUM(?sales) AS ?totalSales)
WHERE {?obs rdf:type qb:Observation ;
          gnw:customerID ?cust ;
          gnw:supplierID ?sup ;
          gnw:salesAmount ?sales .
?cust qb4o:memberOf gnw:customer ;
          gnw:customerGeo ?custGeo .
?sup qb4o:memberOf gnw:supplier;
          gnw:supplierGeo ?supGeo ;
          skos:broader ?supCity .
?supCity qb4o:memberOf gnw:city .
# Inner select for the distance function
  {SELECT ?cust1 (MIN(?distance) AS ?minDistance)
  WHERE {?obs rdf:type qb:Observation ;
          gnw:customerID ?cust1 ;
          gnw:supplierID ?sup1 .
          ?sup1 gnw:supplierGeo ?sup1Geo .
          ?cust1 gnw:customerGeo ?cust1Geo .
          BIND (bif:st_distance(?cust1Geo, ?sup1Geo)
          AS ?distance)}
          GROUP BY ?cust1 }
          FILTER (?cust = ?cust1 && bif:st_distance
          (?custGeo, ?supGeo) = ?minDistance)}
GROUP BY ?supCity ?obs
```

Query D.1: Query with s-roll-up formulated in SPARQL

The fact that data warehouse (DW) queries typically involve nesting of (S)OLAP operators, for example *(s-roll-up(s-slice(s-dice(DW))))*, makes it almost impossible for non-experts to formulate such queries with SPARQL.

Hence, we have developed *GeoSemOLAP*, a framework with an easy-to-use graphical interface that allows non-experts to query spatial semantic data warehouses using high-level SOLAP operators [5].

2 Queries for Spatial Semantic Data Warehouses

To formulate SPARQL queries automatically, GeoSemOLAP needs some information about what data is contained in the spatial data warehouse. Our current implementation uses metadata using the QB4SOLAP vocabulary[1] [4, 5] for this purpose. In addition to MD data elements, QB4SOLAP also describes spatial concepts and builds upon existing vocabularies: *QB*[2] and *QB4OLAP*[3].

A QB4SOLAP data cube contains a set of *observations*. Its structure is defined through a data structure definition (DSD) describing standard concepts, such as *dimensions*, *measures*, *hierarchies*, *hierarchy steps*, *levels*, and *level attributes*, as well as spatial concepts, such as spatial aggregate functions, topological relations, spatial attributes, and spatial measures.

Let us consider an example of a company (Northwind) that exports a number of goods. The company records its sales data[4] in a spatial data warehouse, which is based on an MD model enabling analyses of sales according to their geographical distribution. The company decides to share the data warehouse on the Semantic Web for further analysis of its sales (e.g., involving economic and demographic data published as Open Data on the Web) and to provide access to all branches as well as customers and suppliers. Fig. D.2 illustrates an example schema of the company's spatial data warehouse.

The example query from Section 1 (Query E.1: "Total sales to customers grouped by city of their closest supplier") can on a high level be represented as: S-ROLL-UP (Sales, [DISTANCE(Customer, Supplier)] → ClosestCity, SUM(SalesAmount)). The query's s-roll-up operator takes the sales observations as input. Each sale observation has a set of associated *measures* representing quantitative descriptions, e.g., Sales Amount and Sales Location – the latter corresponds to a *spatial measure*. Measures have *aggregation* functions (e.g., SUM for Sales Amount and Convex Hull for Sales Location) that

[1] QB4SOLAP: `https://w3id.org/qb4solap#`
Vocabulary Structure: `http://extbi.cs.aau.dk/QB4SOLAP`
[2] RDF Data Cube: `https://w3.org/TR/vocab-data-cube/`
[3] QB4OLAP: `https://lorenae.github.io/qb4olap/`
[4] `http://northwinddatabase.codeplex.com/`

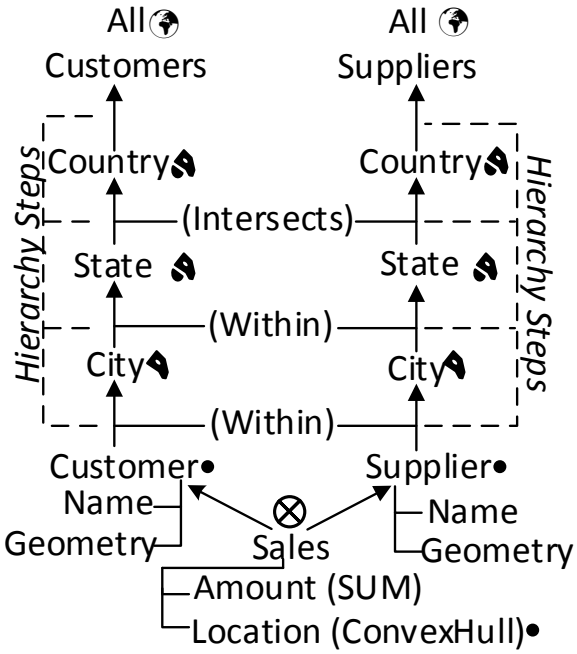


Fig. D.2: Northwind spatial data cube members (symbols next to level names represent spatial characteristics of level members, e.g., point, polygon, and multi-polygon.)

can be used to combine several measure values when rolling-up to a higher *level*, such as from City level to Country level. In Fig. D.2 both Customer and Supplier are *spatial dimension*s.

The graph pattern in Lines 2–11 of Query E.1 describes the roll-up path as a path-shaped join of triple patterns connecting observations to target levels but also to the corresponding measures and attributes.

Line 1 in Query E.1 selects the sales observations (`?obs`) and the *spatial* level (`?supCity`) as output. It also specifies to use aggregation function `SUM` on the measure (`?sales`). Lines 2–5 describe a star-shaped join of triple patterns connecting sales observations (`?obs`) to the Sales Amount measure (`?sales`) as well as to the base level members of two spatial dimensions: Customer (`?cust`) and Supplier (`?sup`).

City→State→Country→ *All* in Fig. D.2 depicts *spatial* hierarchies for the Customer and Supplier dimensions. In Query E.1, Lines 6, 8, and 11 query intermediate spatial levels of these hierarchies. For each spatial level we query the spatial attributes (Lines 7 and 9), e.g., Customer and Supplier have *point*s. Line 10 describes the roll-up from the lowest level in dimension Supplier to

a higher level City. Each roll-up between levels is defined as a hierarchy step. Spatial hierarchy steps have a topological relation between the levels (e.g, Customer$\rightarrow_{(Within)}$City or State$\rightarrow_{(Intersects)}$Country, Fig. D.2), which is annotated with QB4SOLAP in the DSD schema.

The rest of Query E.1 (Lines 12-23) represents an inner select operation for calculating the distances between Customer and Supplier locations, which is necessary to find the closest Supplier cities for the SOLAP operation. The inner select binds the calculated distances to the existing schema members from the outer select. Thus, it is very similar in structure to the first part of the query besides the spatial function (*st_distance*).

3 System Overview

3.1 GeoSemOLAP Workflow

The workflow of querying spatial semantic data warehouses with GeoSemOLAP consists of six main steps, as illustrated in Fig. D.3. First, the user selects a SOLAP operator. In dependence on which SOLAP operator was selected, the user can choose several items from a drop-down menu to complete the definition of the operator, e.g., schema elements (spatial levels, attributes, etc.) and spatial operations (distance, within, etc.).

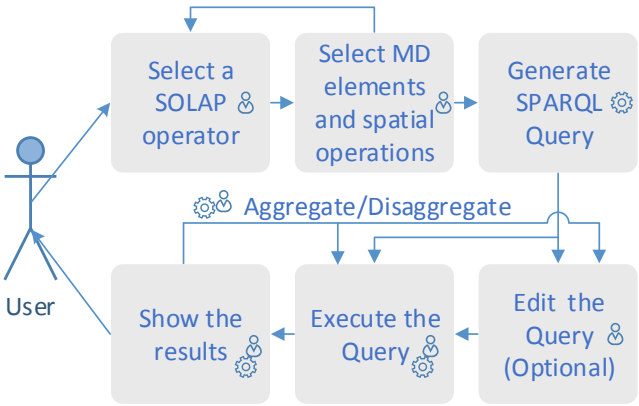


Fig. D.3: Workflow diagram

As some operators (e.g., s-slice, Fig. D.5b) require spatial coordinates as input, GeoSemOLAP displays snippets of geographic maps so that the user can click on a position to indicate spatial coordinates that the query should operate on.

Afterwards, the user can decide to select another SOLAP operator that shall be applied on the results of the previous operator (nesting) and set its parameters.

Then, the SPARQL query is automatically generated by GeoSemOLAP. The query can optionally be edited by the user before GeoSemOLAP sends the query to a SPARQL endpoint (local or remote) for execution.

Finally, the result of the query is displayed to the user. The user may aggregate the results or decide to continue editing the query by for example adding additional SOLAP operators and repeating the steps above mentioned.

3.2 GeoSemOLAP Architecture

GeoSemOLAP consists of five architectural components: *GUI, Metadata Manager, Query Generator, Data Processor*, and *SPARQL Endpoint*. The system architecture is illustrated in Fig. D.4. GeoSemOLAP is implemented in Javascript, HTML, and CSS. It uses Leaflet[5] for visualizing maps and Virtuoso Open Source Edition (7.2) for running the endpoint.

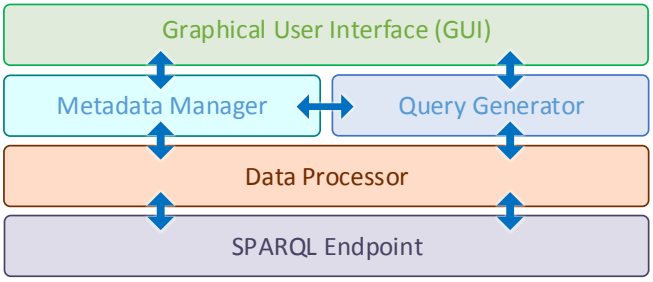


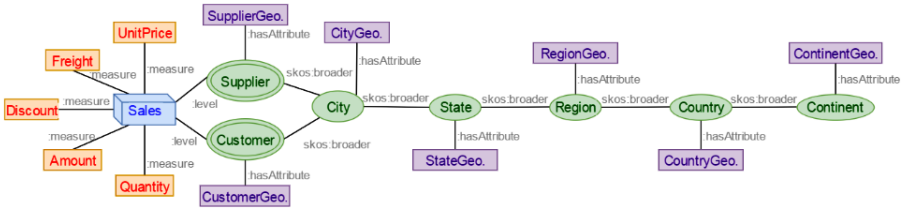
Fig. D.4: GeoSemOLAP architecture

4 Demonstration Scenario

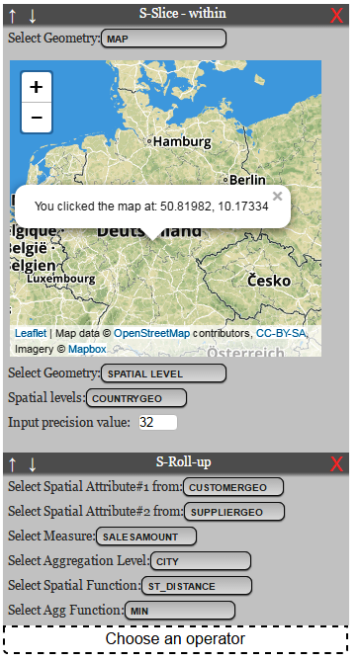
We will demonstrate GeoSemOLAP using two datasets (GeoFarmHerdState [4] and Geo-Northwind), which are both also available at our public endpoint[6]. The Geo-Northwind data cube has already been explained in Sect. 2. GeoFarmHerdState [4] is a spatial data cube about livestock holdings in Denmark. To enable interesting analyses, we have integrated data about livestock holdings with environmental and geographical data.

At the conference, we will demonstrate GeoSemOLAP by allowing attendees to interact with the system and formulate queries. To make it easier for the audience to understand the data and the cube structure, the graphical interface displays a high-level graphical representation at the top of the page – as illustrated in Fig. D.5a.

[5] http://leafletjs.com
[6] http://lod.cs.aau.dk:8890/sparql



(a) Graphical representation of an example use-case schema



(b) SOLAP operator configuration

```
Prefixes
SELECT ?obs2 ?supplierCity ?cityName (SUM(?sales2) AS ?
totalSales) WHERE
{
   ?obs2 rdf:type qb:Observation .
   ?obs2 gnw:customerID ?customer2  .
   ?obs2 gnw:supplierID ?supplier .
   ?customer2  qb4o:memberOf gnw:customer .
   ?customer2  skos:broader ?city2 .
   ?customer2 gnw:customerGeo ?custGeo .
   ?supplier qb4o:memberOf gnw:supplier .
   ?supplier gnw:supplierGeo ?supGeo .
   ?supplier skos:broader ?supplierCity .
   ?supplierCity qb4o:memberOf gnw:city .
   ?supplierCity gnw:cityName ?cityName .
   ?city2  qb4o:memberOf gnw:city .
   ?city2  skos:broader ?state1  .
   ?state1  qb4o:memberOf gnw:state .
   ?state1  skos:broader ?country1  .
   ?country1  qb4o:memberOf gnw:country .
   ?country1  gnw:countryGeo ?countryGeo1  .
   ?obs2 gnw:salesAmount ?sales2 .
{ SELECT ?cust1 (MIN(?distance) AS ?minDistance)
WHERE {
?o2 rdf:type qb:Observation .
?o2 gnw:customerID ?cust1 .
?o2 gnw:supplierID ?sup1 .
?sup1 qb4o:memberOf gnw:supplier .
?sup1 gnw:supplierGeo ?sup1Geo .
?cust1 qb4o:memberOf gnw:customer .
?cust1 gnw:customerGeo ?cust1Geo .
BIND (bif:st_distance( ?cust1Geo, ?sup1Geo ) AS ?distance)}
GROUP BY ?cust1 }
FILTER (?customer2 = ?cust1 && bif:st_distance( ?custGeo, ?supGeo
) = ?minDistance)
FILTER (bif:st_within(bif:st_point(10.173339843750002,
50.819818262156545), ?countryGeo1 ,32))
} GROUP BY ?obs2 ?supplierCity ?cityName
Run Query  Disaggregate  Clear Tables
```

(c) Generated SPARQL query for nested SOLAP

obs2	supplierCity	cityName	totalSales
http://qb4solap.org/cubes/instances/geonorthwind#sale_10643_3	http://qb4solap.org/cubes/instances/geonorthwind#city_22	"Lyngby"	18
http://qb4solap.org/cubes/instances/geonorthwind#sale_10323_2	http://qb4solap.org/cubes/instances/geonorthwind#city_6	"Berlin"	44.8
http://qb4solap.org/cubes/instances/geonorthwind#sale_10312_4	http://qb4solap.org/cubes/instances/geonorthwind#city_14	"Frankfurt"	62

(d) Example result for a nested SOLAP query (S-Roll-up (S-Slice()))

Fig. D.5: Screenshot of GeoSemOLAP

Hence, conference attendees will be able to directly interact with the system, which during the demonstration will run on a laptop connected to an external screen. Although GeoSemOLAP can use a remote endpoint to execute the formulated queries, the demonstration will use a local endpoint (running on the same laptop as the graphical interface) to avoid problems

with an unstable Internet connection.

Fig. D.5 shows a compact screenshot of GeoSemOLAP. At the top, we see a graphical representation of the used data cube. As mentioned above, it shows the most important concepts that are necessary to formulate a SOLAP query, e.g., dimensions, hierarchies, measures, spatial levels, and level attributes.

In this example (Fig. D.5), the user has first selected the operator s-slice. The menu on the left-hand side (Fig. D.5b) allows the user to set the parameters so that the s-slice operator selects geometries from a map or spatial levels from the schema. S-slice requires two spatial parameters; the first one to define a spatial location and the second one to define the slice level with respect to the given location. Hence, to help the user better select coordinates to define a location, a map is displayed so that the user can simply click on a point, which is then automatically extracted. In Fig. D.5b, we can see that the user clicked on a point in Germany and then selected Country from the spatial levels to make a projection on the observations in this country.

In addition, to an s-slice operator, the user has added an s-roll-up operator to aggregate measures and discover new perspectives on sales with respect to their spatial location – the s-roll-up operator with its parameters is displayed under s-slice in Fig. D.5b. The obtained result (from s-roll-up) is similar to our running example introduced in Sect. 1 (Query 1).

Based on the provided operators and parameters, GeoSemOLAP will automatically create and display the corresponding SPARQL query (Fig. D.5c). The user can then decide to run the query and view the results. The results for the example query are displayed at the bottom of the page (Fig. D.5d). We refer to our screencast[7] for a more detailed explanation of GeoSemOLAP.

5 Perspectives and Future Work

Fig. D.6 sketches our vision of a tool-oriented future for SOLAP on the SW based on the models and algorithms proposed in [5].

As mentioned in Sect. 1, there is a growing popularity of spatial LOD datasets from various governmental domains[8,9,10,11]. These datasets contain observations and measures that are well-suited for analytical queries (e.g., water/air quality measurements, immigration rates, EU subsidies in agriculture, crop revenue, etc.). However, as observed in [5] such datasets are typically not modeled with spatial dimension levels and hierarchies. Thus, they cannot directly be queried with SOLAP on the SW.

[7] https://youtu.be/Pc3RBPPgBhA
[8] https://ec.europa.eu/eurostat
[9] https://environment.data.gov.uk
[10] https://datahub.io/dataset/govagribus-denmark
[11] https://datahub.io/dataset/acorn-sat

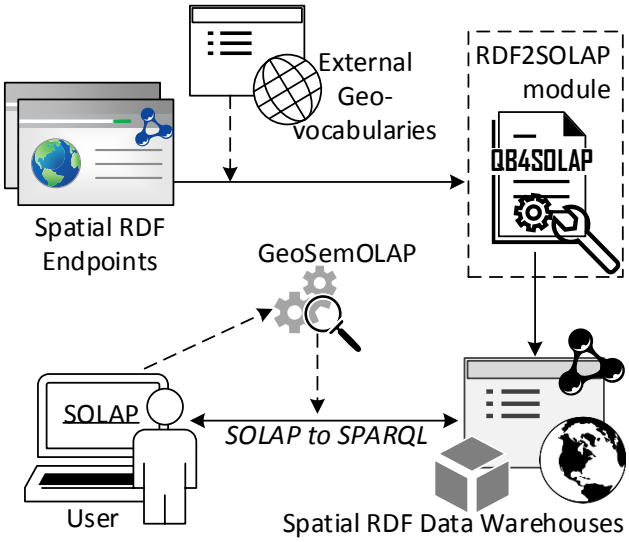


Fig. D.6: Vision of SOLAP on the Semantic Web [5]

With currently available SW technologies, a user, who would like to query available spatial RDF data with SOLAP, needs to download the datasets, map them to a relational data model, and then import the result into a traditional spatial data warehouse. Obviously, this workflow is not only slow but it is also time-consuming and requires storing the data in a non-open format on a local system.

GeoSemOLAP considerably lowers the entry barrier for advanced spatial analysis on the SW by providing a user-friendly interface to formulate SOLAP queries in SPARQL. Our future work strives at lowering the entry barrier even further by developing (semi-)automatic techniques for annotating existing spatial RDF data on the SW with QB4SOLAP and defining spatial levels and hierarchies using available datasets, such as GeoNames[12]. Furthermore, it would be interesting to extend the model proposed in [5] and GeoSemOLAP to handle highly dynamic spatio-temporal data and queries as, for instance, found in large-scale transport analytics [6].

References

[1] M. Koubarakis, M. Karpathiotakis, K. Kyzirakos, C. Nikolaou, and M. Sioutis, "Data Models and Query Languages for Linked Geospatial Data," in *Reasoning Web. Semantic Technologies for Advanced Query*

[12] http://www.geonames.org/

Answering. Springer, 2012, pp. 290–328, https://dx.doi.org/10.1007/978-3-642-33158-9_8.

[2] S. Bimonte, A. Tchounikine, M. Miquel, and F. Pinet, "When spatial analysis meets OLAP: Multidimensional model and operators," *IJDWM*, 2011.

[3] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Semantic Technology: 4th Joint International Semantic Technology Conference (JIST'14)*, vol. 8943. Springer, 2014, pp. 178–186, https://dx.doi.org/10.1007/978-3-319-15615-6_13.

[4] N. Gür, K. Hose, T. B. Pedersen, and E. Zimányi, "Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP," in *Semantic Technology: 6th Joint International Semantic Technology Conference (JIST'16)*, vol. 10055. Springer, 2016, pp. 287–304, https://dx.doi.org/10.1007/978-3-319-50112-3_22.

[5] N. Gür, T. B. Pedersen, E. Zimányi, and K. Hose, "A Foundation for Spatial Data Warehouses on the Semantic Web," *Semantic Web Journal*, vol. 9, no. 5, pp. 557–587, 2018.

[6] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler, "Highly scalable trip grouping for large-scale collective transportation systems," in *EDBT*, 2008.

Acknowledgements

This research is partially funded by the European Commission through the Erasmus Mundus Joint Doctorate Information Technologies for Business Intelligence (EM IT4BI-DC) and the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301.

Paper E

Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP

Nurefşan Gür, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi

The paper has been published in the
Proceedings of the 6th Joint International Semantic Technology Conference
Vol. 10055, pp. 287–304, 2016. DOI: 10.1007/978-3-319-50112-3_22

Abstract

Governmental organizations and agencies have been making large amounts of spatial data available on the Semantic Web (SW). However, we still lack efficient techniques for analyzing such large amounts of data as we know them from relational database systems, e.g., multidimensional (MD) data warehouses and On-line Analytical Processing (OLAP). A basic prerequisite to enable such advanced analytics is a well-defined schema, which can be defined using the QB4SOLAP vocabulary that provides sufficient context for spatial OLAP (SOLAP). In this paper, we address the challenging problem of MD querying with SOLAP operations on the SW by applying QB4SOLAP to a non-trivial spatial use case based on real-world open governmental data sets across various spatial domains. We describe the process of combining, interpreting, and publishing disparate spatial data sets as a spatial data cube on the SW and show how to query it with SOLAP operators.

Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP. In: *Semantic Technology*, JIST 2016. Lecture Notes in Computer Science.
https://doi.org/10.1007/978-3-319-50112-3_22
The layout has been revised.

1 Introduction

In late 2012, the Danish government joined the Open Data movement by making several raw digital data sets [1] freely available at no charge. These data sets span domains such as environmental data, geospatial data, business data from transport to tourism, fishery, forestry, and agriculture. GovAgriBus Denmark[1] was an initial effort in 2014 to make Danish government Open Data from various domains available as Linked Open Data (LOD) [2] on the Semantic Web in order to pose queries across domains. If the corresponding domains can be related through space and location, spatial attributes of these data sets become particularly interesting as we can derive spatial joins and containment relationships that were not encoded in the original data sets. Danish government organizations and agencies continue publishing data sets for new domains and update the corresponding data sets regularly on a yearly basis, which brings opportunities in querying the expanding spatial data with analytical perspectives on the Semantic Web. Responding to such queries is a complex task, which requires well-defined schemas to facilitate OLAP operations on the Semantic Web. QB4SOLAP [3] aims to support intelligent multidimensional querying in SPARQL by providing context to SOLAP and its elements on the SW. However, QB4SOLAP has not been applied on complex real-world data yet. This could bring particular challenges with the use of real-world spatial data. In this paper, we address the challenging problem of multidimensional querying with SOLAP operations on the SW by applying QB4SOLAP on real-world open governmental data sets from various domains. These domains span from livestock farming to environment, where many of them have spatial information.

In this paper, we design a spatial data cube schema with data from livestock farming, environment, and geographical domains. Every data set is downloaded from different governmental sources in various formats. The downloaded data is prepared and conciliated with a spatial data cube schema in order to publish it on the SW with QB4SOLAP. We use the common SOLAP operators [4] on the spatial data cube for advanced analytical queries. These analytical queries give perspective on the use case data sets that are linked and published with QB4SOLAP as a unified spatial data cube. Having the use case data sets with spatial attributes also allows us to reveal patterns across the use case domains that were not possible before. We share our experiences with the best practices and methods together with the lessons learned. Finally, we show how to formulate and execute SPARQL queries with individual and nested SOLAP operations.

The remainder of this paper is structured as follows. Section 2 presents the background and motivation, Section 3 discusses related work and presents

[1] `https://datahub.io/dataset/govagribus-denmark`

the state-of-the-art spatial data cubes on the SW. Section 4 presents the data sources for the use case while Section 5 describes how to annotate and publish the use case data as a spatial data cube on the SW. Section 6 presents SOLAP operators and their SPARQL implementation. Section 7 presents a brief overview of the process and reflects on the problems and improvements. Finally, Section 8 concludes the paper with an outlook to future work.

2 Background and Motivation

The Semantic Web supports intelligent querying via SPARQL with active inference and reasoning on the data in addition to capturing its semantics. Linked Open Data on the Semantic Web is an important source to support Business Intelligence (BI). Multidimensional data warehouses and OLAP are advanced analytical tools in analyzing complex BI data. State-of-the-art SW technologies support advanced analytics over *non-spatial* SW data. QB4SOLAP supports intelligent multidimensional querying in SPARQL by providing context to spatial data warehouses and its concepts. Variety of the data is an intriguing concept on both the Semantic Web and in complex BI systems. The variety of the data and heterogeneous representation formats (e.g., CSV, JSON, PDF, XML, and SHP) require underlying conceptualizations and data models with well-defined spatial (and temporal) dependencies, which can be modeled with QB4SOLAP in order to answer complex analytical queries. Complex queries cannot be answered from within one domain alone but span over multiple disciplines and various data sources. As a result, this paper is driven by the motivation of using QB4SOLAP as a proof of concept for spatial data warehouses on the Semantic Web by using open (government) data of various domains from different sources, which creates a non-trivial spatial use case.

3 State of the Art

Data warehouses and OLAP technologies have been successful for analyzing large volumes of data [5], including integrating with external data such as XML [6]. Combining DW/OLAP technologies with RDF data makes RDF data sources more available for interactive analysis. Kämpgen et al. propose an extended model [7] on top of the RDF Data Cube Vocabulary (QB) [8] for interacting with statistical linked data via OLAP operations directly in SPARQL. In OLAP4LD [9], Kämpgen *et al.* suggest enhancing query performance of OLAP operations expressed as SPARQL queries by using RDF aggregate views. The W3C published a list of RDF cube implementations [10]. However, they all have inherent limitations of QB and thus cannot support

OLAP dimensions with hierarchies and levels, and built-in aggregate functions.

Etcheverry et al. [11] introduce QB4OLAP as an extended vocabulary based on QB, with a full MD metamodel, supporting OLAP operations directly over RDF data with SPARQL queries. Matei *et al.* [12] use QB and QB4OLAP as a basis to support OLAP queries in Graph Cube [13] with the IGOLAP vocabulary. Jakobsen *et al.* [14] study OLAP query optimization techniques over QB4OLAP data cubes. However, none of these approaches and vocabularies support *spatial* DWs.

QB4SOLAP(v1) [3] is the first attempt to model and query spatial DWs on the SW, and QB4SOLAP(v2) [4] is a foundation for spatial data warehouses and SOLAP operators on the SW, which is currently under submission with completely revised formal semantics of SOLAP operators and SPARQL query generations algorithms. QB4SOLAP is an extension of QB4OLAP with spatial concepts. The QB4OLAP vocabulary is compatible with the QB vocabulary. Therefore QB4SOLAP provides backward compatibility with other statistical or MD data cube vocabularies in addition to providing spatial context for querying with SOLAP. Fig. E.1 depicts the QB4SOLAP(v2) vocabulary. Capitalized terms with non-italic font represent RDF classes, capitalized terms with italic font represent RDF instances, and non-capitalized terms represent RDF properties. Classes in external vocabularies are depicted in light gray background and font. QB, QB4OLAP, and QB4SOLAP classes are shown with white, light gray, and dark gray backgrounds. Original QB terms are prefixed with `qb:`[2]. QB4OLAP and QB4SOLAP terms are prefixed with `qb4o:`[3] and `qb4so:`[4] . Spatial classes are prefixed with `geo:`[5], where the spatial extension to QB4SOLAP is based on the GeoSPARQL [15] standard from the Open Geospatial Consortium (OGC) for representing and querying geospatial linked data on the SW.

QB4SOLAP is a promising approach for modeling, publishing, and querying spatial data warehouses on the SW. However, it has only been validated with a synthetic use case. Andersen et al. [2] consider publishing/converting open Danish governmental spatial data as Linked Open Data without considering the MD aspects of geospatial data. In this paper, however, we validate QB4SOLAP with a non-trivial use case, which is created as a spatial data cube from open Danish government spatial data. Furthermore, we show how to exploit multidimensional spatial linked data on the SW, which is not solely about adding semantics and linking disparate data sets on the SW, but also about enabling analytical queries by modeling them as spatial data cubes.

[2]RDF Cube: `http://purl.org/linked-data/cube#`
[3]QB4OLAP: `http://purl.org/qb4olap/cubes#`
[4]QB4SOLAP: `http://w3id.org/qb4solap#`
[5]GeoSPARQL: `http://www.opengis.net/ont/geosparql#`

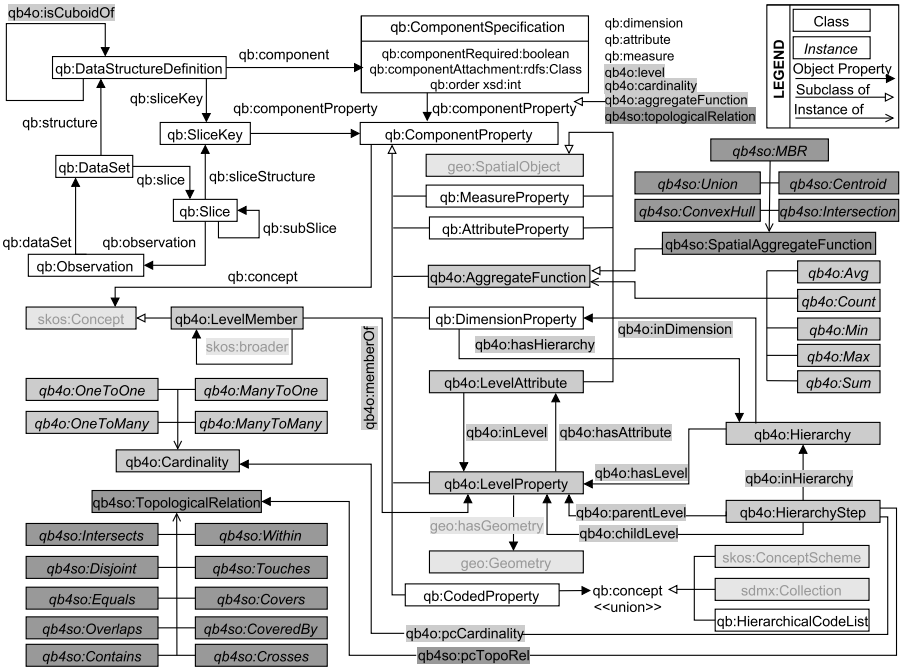


Fig. E.1: QB4SOLAP Vocabulary

4 Source Data

In order to create a spatial data cube of livestock holdings in Danish farms, we have gathered data that is published by different agencies in Denmark. We have found these domains to be particularly interesting as they represent a non-trivial use case that covers spatial attributes and measures, which can be modeled in a spatial data cube for multidimensional analysis. In the following we first give a brief overview of the flat data and their sources, and then represent the whole use case data set as a spatial data cube in Section 5.

The environmental protection agency under the Ministry of Environment and Food of Denmark regulates the livestock units (DE)[6] per area in order to keep nitrate leaching under control in vulnerable areas. Prohibition rules against the establishment of livestock farms and the siting of animal housing are determined with respect to livestock units and distance to specific natural habitats (e.g., ammonia vulnerable areas, water courses, and water supply facilities etc.) [16].

[6]Livestock units are used to produce statistics describing the number of livestocks in farms.

Livestock Farming (CHR) Data.

The Ministry of Environment and Food of Denmark (`http://en.mfvm.dk`) publishes the central husbandry (livestock) registry (CHR) data, which is the central database used for registration of holdings and animals. We refer to this set of data as *CHR data*. We have downloaded several relevant data sets from `http://jordbrugsanalyser.dk` in livestock farming domain. The CHR data collection is downloaded in SHP format as 6 data sets, where each data set represents the state of the farms for a year between the years 2010 to 2015. SHP format is used for *shapefile*s, which store geometry information of the spatial features in a data set. In each shapefile, there is information about more than 40,000 farms. In total, the CHR data collection contains around 240,000 records. Farm locations are given as `(X,Y)` point coordinates. Each data set has 24 attributes in which the important ones are: *CHR - Central Husbandry (animal) Registry (holding) number, CVR - Central Company Registry number (owner company of the holding), DE (Livestock unit), Address of the holding (Postnr and Commune), Geographical position of the holding (X and Y coordinates), Different type of normalized herds, Number of animals for each herd, Animal code and label, Animal usage code and purpose.*

Environmental Data.

Public environmental data is published on Denmark's environment portal `http://www.miljoeportal.dk/`, where we can find information about nitrate catchment areas and vulnerable sites. The soil measurements contain data from 2008 to 2015 [17]. We downloaded the data sets in SHP format, which have recently become available on the portal. The files record measurements of the soil quality across Denmark. The environmental data collection contains 3 data sets about nitrogen reduction potentials and phosphor and nitrate classifications of the soil. Temporal validity of the soil measurement data is recorded in the attributes with timestamps. Each data set keeps records of polygon areas. In total, the environmental data collection contains around 30,000 records. Datasets have attribute fields about the area of the polygons, CVR number of the data provider agency or company, responsible person name, etc. The important attributes, which record the soil measurement data are: *Nitrate class type, Nitrogen reduction potentials, Phosphor class type*.

Geographical (Regions) Data.

The primary use case data is built around livestock farming (CHR) and environmental data as mentioned above. In order to pursue richer analysis upon this use case we enrich the spatiality of the use case data by adding two geographical data sets; parishes and drainage areas of Denmark. These data sets are spatially and topically relevant since we have

found pre-aggregated maps created by the Ministry of Environment and Food of Denmark at parish and drainage area levels for livestock farming data. We downloaded parishes and drainage areas of Denmark as SHP files from `http://www.geodata-info.dk/`. The total number of records of the geographical data collection are 2,300. These data sets have attribute fields such as: *Drainage area name, Parish name, Total area, etc.*

Central Company Registry (CVR) data.

Danish companies, agencies and industries are registered in the Central Company Register (CVR). Every livestock holding is owned by a company and has a CVR number. Environmental data also records the CVR number of the corresponding data provider agencies. Through this CVR number, we can access detailed information of the companies and contact details of the responsible people. This collection allows evaluating interesting queries with the selected domains given above. The CVR data is published at `http://cvr.dk` and can be accessed via a web service with a Danish social security number log-in. We accessed and downloaded only the data in CSV format that are accredited for publishing. This data includes attributes such as: *Company name, Phone number, and Address etc.*

5 Publishing Spatial Data Cubes with QB4SOLAP

The QB4SOLAP vocabulary allows to define *cube schemas* and *cube instances*. A cube schema defines the structure of a cube as an instance of the class `qb:DataStructureDefinition` in terms of dimension levels, measures, aggregation functions (e.g., SUM, AVG, and COUNT) on measures, spatial aggregation functions[7] on spatial measures, fact-level cardinality relationships, and topological relationships. The properties used to express these relationships are: `qb4o:level`, `qb:measure`, `qb4o:aggregateFunction`[8], `qb4o:cardinality`, and `qb4so:topologicalRelation` respectively (Fig. E.1). These schema level metadata are used to define MD data sets in RDF. Cube instances are the members of a cube schema that represent level members, facts and measure values. We describe the cube schema elements in Section 5.1 and the cube instances in Section 5.2 with their examples.

[7]Spatial aggregation functions aggregate two or more spatial objects and return a new spatial object, e.g., union, buffer, and convexHull etc.

[8]SpatialAggregateFunction is a subclass of AggregateFunction. Thus, measures and spatial measures use the same property `qb4o:aggregateFunction`.

5.1 GeoFarmHerdState Cube Schema in RDF

As our use case we create a spatial data cube of livestock holdings that we refer to as *GeoFarmHerdState*. The use case data cube is created from the flat data sets of the livestock farming (CHR) data, the environmental data, the geographical (regions) data, and the company registry (CVR) data collections, which are explained in Section 4. We create this data cube by thoroughly analyzing the attributes of the flat data sets from the collected relevant domains and conciliating them by foreign keys or by overlaying the SHP files of the spatial data sets and deriving new attributes from the intersected areas. After deriving useful spatial information across use case domains, we generalize the tabular data of the several use case data sets into the GeoFarmHerdState data cube. Fig. E.2 shows the multidimensional conceptual schema of the GeoFarmHerdState spatial cube. The multidimensional elements of the cube are explained in Remarks 1 – 7 followed by their examples in RDF. The underlying syntax for RDF examples is given in Turtle. We prefix the schema elements of the GeoFarmHerdState cube with `gfs:`.

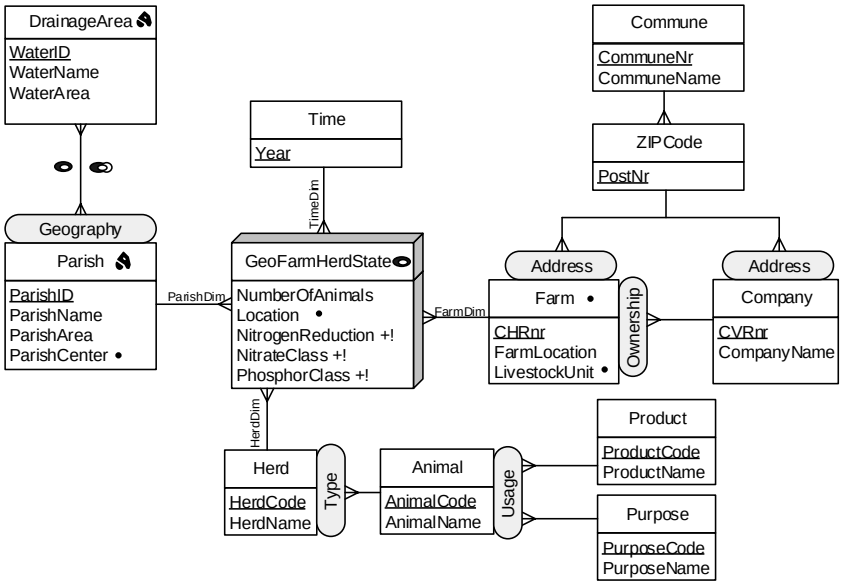


Fig. E.2: GeoFarmHerdState – Conceptual MD schema of livestock holdings data

Remark 1. (Dimensions) *Dimensions* provide perspectives to analyze the data. The GeoFarmHerdState cube has four dimensions, in which the two of them are *spatial* (FarmDim, ParishDim). All dimensions in the cube are defined with `qb:DimensionProperty`. A dimension is spatial if it has at least one spatial level (See Remark 3). Dimension hierarchies are defined with `qb4o:hasHierarchy` property. Hierarchies and their types are explained later in Remark

2.

Example 1 (Dimensions)
We give two spatial dimensions as an example.

```
gfs:farmDim rdf:type qb:DimensionProperty ; qb4o:hasHierarchy gfs:ownership , gfs:address .
gfs:parishDim rdf:type qb:dimensionProperty ; qb4o:hasHierarchy gnw:geography .
```

Remark 2. (Hierarchies) *Hierarchies* allow users to aggregate measures at various levels of detail. Hierarchies are composed of levels. A hierarchy is *spatial* if it has at least one spatial level (See Remark 3). Hierarchies of the GeoFarmHerdState cube are given in ellipses (Fig. E.2). Each hierarchy is defined with `qb4o:Hierarchy` and linked to its dimension with the `qb4o:inDimension` property. Levels that belong to the hierarchy are defined with the `qb4o:hasLevel` property.

Example 2 (Hierarchies)
We present the most interesting hierarchies from the GeoFarmHerdState cube as an example.

```
gfs:geogprahy rdf:type qb4o:Hierarchy ; qb4o:inDimension gfs:parishDim ;
    qb4o:hasLevel gfs:drainageArea .
gfs:usage rdf:type qb4o:Hierarchy ; qb4o:inDimension gfs:animalDim ;
    qb4o:hasLevel gfs:product , gfs:purpose .
gfs:address rdf:type qb4o:Hierarchy ; qb4o:inDimension gfs:farmDim ;
    qb4o:hasLevel gfs:zipCode , gfs:commune .
```

The Geography hierarchy is a *non-strict* spatial hierarchy. A spatial hieachy is non-strict if it has at least one $(n - n)$ relationship between its levels. In the Geography hierarchy (Fig. E.2) the $(n - n)$ cardinality represents that a parish may belong to more than one drainage area. Usually, non-strict spatial hierarchies arise when a partial containment relationship exists, which is given as *Intersects* in our use case. Usage hierarchy is a *generalized* hierarchy with non-exclusive paths to splitting levels (Product and Purpose) and has no joining level but the top level *All*. Finally, the Address and Ownership hierarchies are *parallel dependent* hierarchies. Parallel hierarchies arise when a dimension has several hierarchies sharing some levels. Note that the Address hierarchy has different paths from the Company and Farm levels (Fig. E.2).

Remark 3. (Levels) *Levels* have a set of attributes (See Remark 4) that describes the characteristics of the level members (See Remark 9). Levels are defined with the `qb4o:LevelProperty` and their attributes are linked with

the qb4o:hasAttribute property. A level is *spatial* if it has an associated geometry. Therefore, spatial levels have the property geo:hasGeometry, which defines the geometry of the spatial level in QB4SOLAP.

Example 3 (Levels and Level attributes)
We present a spatial level (Parish) as an example with its attributes. Attributes and spatial attributes of levels are further described in Remark 4.

```
gfs:parish rdf:type qb4o:LevelProperty ; qb4o:hasAttribute gfs:parishID ;
    qb4o:hasAttribute gfs:parishName ; qb4o:hasAttribute gfs:parishArea ;
    qb4o:hasAttribute gfs:parishCenter ; geo:hasGeometry gfs:parishPolygon.
```

Note that the Parish level is defined as a spatial level because it has an associated polygon geometry (gfs:parishPolygon), which is specified with the geo:hasGeometry property. Some other spatial characteristics of the levels can be recorded in the spatial attributes of the level such as the center point of the parish (gfs:parishCenter).

Remark 4. (Attributes) Attributes and *spatial* attributes are defined with the qb4o:LevelAttribute property and linked to their levels with the qb4o:inLevel property. An attribute is spatial if it is defined over a spatial domain. Attributes are defined as ranging over XSD literals[9] and spatial attributes must be ranging over spatial literals, i.e., well-known text literals (WKT) from OGC schemas[10]. Spatial attributes are a sub-property of the geo:Geometry class. Further, the domain of the spatial attribute should be specified with rdfs:domain, which must be a geometry. Finally, the spatial attribute must be specified as an instance of geo:SpatialObject with the rdfs:subClassOf property. Examples of attributes are given in the following.

Example 4 (Spatial and non-spatial attributes)
We present spatial and some non-spatial attributes of the Parish level.

```
gfs:parishID rdf:type qb4o:LevelAttribute ; qb4o:inLevel gfs:parish ;
    rdfs:range xsd:positiveInteger .
gfs:parishName rdf:type qb4o:LevelAttribute ; qb4o:inLevel gfs:parish ;
    rdfs:range xsd:string .
gfs:parishCenter rdf:type qb4o:LevelAttribute ; rdfs:subPropertyOf geo:Geometry ;
    qb4o:inLevel gfs:parish ; rdfs:domain geo:Point; rdfs:subClassOf geo:SpatialObject;
    rdfs:range geo:wktLiteral , virtrdf:Geometry .
```

We have mentioned in Remark 3 that spatial levels are defined through their associated geometries, which are not given as a level attribute. For the Parish level we present the following example of the corresponding geometry.

[9] XML Schema Definition: http://www.w3.org/TR/xmlschema11-1/
[10] OGC Schemas: http://schemas.opengis.net/

```
gfs:parishPolygon rdf:type geo:Geometry; rdfs:domain geo:MultiSurface;
    rdfs:subClassOf geo:SpatialObject; rdfs:range geo:wktLiteral , virtrdf:Geometry .
```

Remark 5. (Hierarchy Steps) Hierarchy steps define the structure of the hierarchy in relation to its corresponding, levels. A hierarchy step entails a roll-up relation between a lower (child) level and an upper (parent) level with a cardinality. The cardinality $(n-n, 1-n, n-1, n-n)$ relationship describes the number of members in one level that can be related to a member in the other level for both child and parent levels. A hierarchy step is *spatial* if it relates a spatial child level and a spatial parent level, in which case it entails a topological relationship between these spatial levels. Both spatial and non-spatial hierarchy steps are defined as a blank node with the `qb4o:HierarchyStep` property and linked to their hierarchies with the `qb4o:inHierarchy` property. The parent and child levels are linked to hierarchy steps with the `qb4o:childLevel` property and the `qb4o:parentLevel` property. The cardinality of a hierarchy step is defined by the `qb4o:pcCardinality` property. And finally, the topological relationship[11] of a hierarchy step is defined by the `qb4so:pcTopoRel` property.

Example 5 (Spatial Hierarchy steps)
The following illustrates the hierarchy steps of the spatial hierarchy Geography and non-spatial hierarchy Address as it has different paths from child levels Farm and Company.

```
## Geography hierarchy structure ##
_:geography_hs1 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:geography ;
    qb4o:childLevel gfs:parish ; qb4o:parentLevel gfs:drainageArea ;
    qb4o:pcCardinality qb4o:ManyToMany; qb4so:pcTopoRel qb4so:Intersects, qb4so:Within .
## Address hierarchy structure ##
_:farm_address_hs1 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:farm ; qb4o:parentLevel gfs:zipCode ;
    qb4o:pcCardinality qb4o:ManyToOne .
_:farm_address_hs2 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:zipCode ; qb4o:parentLevel gfs:commune ;
    qb4o:pcCardinality qb4o:ManyToOne .
_:company_address_hs1 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:company ; qb4o:parentLevel gfs:zipCode ;
    qb4o:pcCardinality qb4o:ManyToOne .
_:company_address_hs2 rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:address ;
    qb4o:childLevel gfs:zipCode ; qb4o:parentLevel gfs:commune ;
    qb4o:pcCardinality qb4o:ManyToOne .
```

Remark 6. (Measures) Measures record the values of a phenomena being observed. Measures and *spatial* measures are defined with `qb:MeasureProperty`.

[11] Topological relations are Boolean predicates that specify how two spatial objects are related to each other, e.g., within, intersects, touches, and crosses etc.

A measure is spatial if it is defined over a spatial domain. Similarly to attributes (Remark 4), measures are defined ranging over XSD literals and spatial measures must be ranging over spatial literals.

Example 6 (Spatial and non-spatial measures)
The following shows an example of a spatial measure (Location) and a non-spatial measure (NumberOfAnimals).

```
gfs:location rdf:type qb:MeasureProperty ; rdfs:subPropertyOf sdmx-measure:obsValue ;
    rdfs:subClassOf geo:SpatialObject ; rdfs:domain geo:Point ;
    rdfs:range geo:wktLiteral , virtrdf:Geometry .
gfs:numberOfAnimals rdf:type qb:MeasureProperty ;
    rdfs:subPropertyOf sdmx-measure:obsValue ; rdfs:range xsd:decimal .
```

Remark 7. (Fact) Fact defines the data structure (DSD) of the cube with `qb:DataStructureDefinition`. The dimensions are given as `components` and defined with the `qb4o:level` property as the dimensions are linked to the fact at the lowest granularity level. A fact is *spatial* if it relates two ore more spatial levels. Similarly, measures are given as `components` of the fact and are defined with the `qb:measure` property. Aggregation functions on measures and spatial aggregation functions on spatial measures are also defined in the DSD with `qb4o:aggregateFunction`. Fact-level cardinality relationships and topological relationships are defined with `qb4o:cardinality` and `qb4so:topologicalRelation` in DSD, respectively (Fig. E.1).

Example 7 (Fact schema)
The following shows the data structure definition of the cube GeoFarmHerdState, which is defined with corresponding measures and dimensions.

```
# – GeoFarmHerdState Cube Definition of the Fact FarmHerdState
gfs:GeoFarmHerdState rdf:type qb:DataStructureDefinition ;
    # Lowest level for each dimensions in the cube
    qb:component [qb4o:level gfs:herd ; qb4o:cardinality qb4o:ManyToOne ] ;
    qb:component [qb4o:level gfs:time ; qb4o:cardinality qb4o:ManyToOne ] ;
    qb:component [qb4o:level gfs:farm ; qb4o:cardinality qb4o:ManyToOne ;
        qb4so:topologicalRelation qb4so:Equals ] ;
    qb:component [qb4o:level gfs:parish ; qb4o:cardinality qb4o:ManyToMany ;
        qb4so:topologicalRelation qb4so:Within ] ;
    # Measures in the cube
    qb:component [qb:measure gfs:numberOfAnimals ; qb4o:aggregateFunction qb4o:Sum] ;
    qb:component [qb:measure gfs:location ; qb4o:aggregateFunction qb4so:ConvexHull] ;
    qb:component [qb:measure gfs:nitrogenReduction ; qb4o:aggregateFunction qb4o:Avg] ;
    qb:component [qb:measure gfs:nitrateClass ; qb4o:aggregateFunction qb4o:Avg] ;
    qb:component [qb:measure gfs:phosphorClass ; qb4o:aggregateFunction qb4o:Avg] .
```

5.2 GeoFarmHerdState Cube Instances in RDF

Cube instances are the members of a cube schema that represent level members and facts (members), which are explained in Remarks 8 and 9 below. We prefix the instances of the GeoFarmHerdState cube with `gfsi:`.

Remark 8. (Fact members) Fact members (i.e., facts of FarmHerdState) are instances of the `qb:Observation` class. Each fact member is related to a set of dimension *base* level members and has a set of measure values. Every fact member has a unique identifier (IRI) which is prefixed with `gfsi:`.

Example 8 (Fact members)
The following shows an example of a single fact member, which represents the state of a farm with CHR no. 39679 in the year 2015 that has the herd code 15.

```
gfsi:farm_39679_2015 rdf:type qb:Observation ;
## Dimension levels and base level members associated with the fact member
    gfs:herdCode gfsi:herd_15 ; gfs:year gfsi:year_2015 ;
    gfs:chrNumber gfsi:farm_39679 ; gfs:parishID gfsi:parish_8311 ;
## Measures associated with the fact member
    gfs:numberOfAnimals "100.0"^^xsd:decimal ; gfs:nitrateClass "3"^^xsd:integer ;
    gfs:nitrogenReduction "0.75"^^xsd:decimal ; gfs:phosporClass "3"^^xsd:integer ;
    gfs:location "POINT(8.3713 56.7912)"^^geo:wktLiteral .
```

Remark 9. (Level Members) Level members are defined in `qb4o:LevelMember` class. They are linked to their corresponding levels from the schema with the `qb4o:memberOf` property. For each level member there is a set of attribute values. Due to the roll-up relations between levels of hierarchy steps (Remark 5),the `skos:broader` property relates a child level member to its parent level member.

Example 9 (Level Members)
The following shows an example of a child level member in the Parish level and one of its parent level members in the DrainageArea level from the Geography dimension. Fig. E.3 presents a map snapshot for fact members and level members. Parish level member “Astrup" is highlighted and DrainageArea level member “Mariager Inderfjord" is marked with red borders. Note that Astrup intersects another drainage area “Langerak", therefore it links to two parent level members via `skos:broader`.

```
## Parish level member
gfsi:parish_8648 rdf:type gfs:parish ;
    qb4o:memberOf gfs:parish ; skos:broader gfsi:water_3710, gfsi:water_159 ;
    gfs:parishID 8311 ; gfs:parishName "Astrup" ; gfs:parishArea 46,118 ;
    gfs:parishCenter "POINT(8.2552, 56.8176)"^^geo:wktLiteral ;
    gfs:parishPolygon "POLYGON((8.4038 56.7963, 8.3984 56.7721, 8.3411 56.7372, 8.3078
```

```
    56.7281, 8.2987 56.7601, 8.2563 56.7763, 8.3511 56.8137, 8.4038 56.7963))"^^geo:wktLiteral .
## DrainageArea level member
gfsi:water_159 rdf:type gfs:drainageArea ;
    qb4o:memberOf gfs:drainageArea ; gfs:waterID 159 ;
    gfs:waterName "Mariager Inderfjord" ; gfs:waterArea 267,477 ;
    gfs:drainageGeo "POLYGON((8.6048 56.9843, 8.5908 56.8969, 8.5707 56.8664,
    8.5975 56.8519, 8.5215 56.8483, 8.3959 56.7625, 8.3938, 56.7340, 8.3613 56.6802,
    8.2584 56.7764, 8.2475 56.7051, 8.2175 56.7232, 8.3121 56.8441, 8.2806 56.8659,
    8.3602 56.9569, 8.4786 56.9713, 8.5474 56.9905, 8.6048 56.9843))"^^geo:wktLiteral .
```

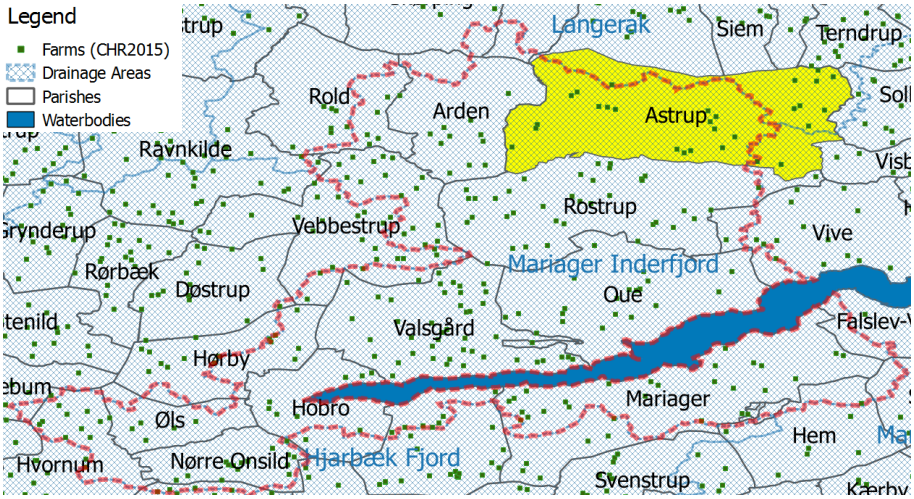


Fig. E.3: GeoFarmHerdState – Fact members and Level members of Ex. 9 marked

6 SOLAP Operators over GeoFarmHerdState cube

Spatial OLAP (SOLAP) operates on spatial data cubes. SOLAP increases the analytical capabilities of OLAP by taking into account the spatial information in the cube. SOLAP operators involve spatial conditions or spatial functions. Spatial conditions specify constraints (i.e., spatial Boolean predicates) on the geometries associated to cube members or measures, while spatial functions derive new data from the cube, which can be used, e.g., to derive dynamic spatial hierarchies.

6.1 SOLAP operators

In what follows we present common SOLAP operators and examples of these operators on GeoFarmHerdState cube.

Remark 10. (S-Slice) The s-slice operator removes a dimension from a cube by choosing a single spatial value in a spatial level. It returns a cube with one dimension less.

Example 10 (S-Slice)
We can perform an s-slice operation in different ways.
1. Slice on farms (state of the farms) of the largest parish.
2. Slice on farms (state of the farms) of the drainage area containing `"POINT(10.43951 55.47006)"`.

The first one applies a spatial function call (for finding the *largest* parish by area) on a spatial level Parish and performs the slice. The second one applies a spatial predicate (for finding where a given point is *within* a particular drainage area) in a spatial level DrainageArea and performs the operation. The corresponding SPARQL queries are:

```
# 1 - s-slice with spatial function #
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gfs:parishID ?parish .
   ?parish gfs:parishPolygon ?parishGeo .
# Inner select for finding the largest parish
   { SELECT ?x (MAX(?area) as ?maxArea) WHERE{
   ?obs rdf:type qb:Observation ;
      gfs:parishID ?parish .
   ?parish gfs:parishPolygon ?x .
   BIND (bif:st_area(?x) as ?area)}}
FILTER ?parishGeo = ?x) }
```

```
# 2 - s-slice with spatial predicate #
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gfs:parishID ?parish .
   ?parish qb4o:memberOf gfs:parish ;
      skos:broader ?drainageArea .
   ?drainageArea gfs:drainageGeo ?drainageGeo .
FILTER (bif:st_within("POINT(10.43951 55.47006)",
?drainageGeo)) }
```

Remark 11. (S-Dice) The s-dice operator keeps the cells of the cube that satisfy the spatial predicate over dimension levels, attributes, or measures. It returns a subset of the cube with filtered members of the cube.

Example 11 (S-Dice)
In the following we show two examples of the s-dice operator.
1. Filter the farms located within 5 km buffer from the center of a drainage area.
2. Filter the farms located within 2 km distance from the center of their parish, which are in the nitrate class I areas.

In the first s-dice operation, initially, a spatial function is applied on level members of the DrainageArea level to get the *center* of their polygon geometries. Then, the level members of the Farm level are filtered with a spatial Boolean predicate with respect to the farm locations that are *within* a 5 km buffer area of the center of the drainage areas. In the second s-dice operation, a spatial function is applied to the spatial measure farm location to get the *distance* of the farms from the center of their parish, which is followed by Boolean predicates; to filter the farms that are less than 2 km away from the center of their parishes and are on nitrate class I areas.

```
# 1 - s-dice on dimension levels #
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gfs:farmID ?farm ;
      gfs:parishID ?parish .
   ?farm gfs:farmLocation ?farmGeo .
   ?parish qb4o:memberOf gfs:parish ;
      skos:broader ?drainageArea .
   ?drainageArea gfs:waterPolygon ?drainagePoly .
BIND (bif:st_centroid (?drainagePoly) as ?drainageCenter)
FILTER (bif:st_within(?drainageCenter, ?farmGeo, 5)) }
```

```
# 2 - s-dice on measures #
SELECT ?obs WHERE {
   ?obs rdf:type qb:Observation ;
      gfs:location ?farmLocation ;
      gfs:nitrateClass ?nitClass ;
      gfs:parishID ?parish.
   ?parish gfs:parishCenter ?parishCent .
BIND (bif:st_distance (?farmLocation, ?parishCent)
AS ?distance)
FILTER (?distance < 2 && ?nitClass = 1)}
```

Remark 12. (S-Roll-up) The s-roll-up operator aggregates measures of a given cube by using an aggregate function and a spatial function along a spatial di-

mension's hierarchy. It returns a cube with measures at a coarser granularity for a given dimension.

Example 12 (S-Roll-up)
In the following, we present two examples of the s-roll-up operator.
1. Total amount of animals in the farms, which are closest to their parishes' center.
2. Average percentage of nitrogen reduction potentials in the parishes that are within and/or intersect the drainage area "Nibe-Bredning".

In the first s-roll-up operator, measures are aggregated to the Parish level after selecting the farms with respect to their proximity to the center of the parish with a spatial function. In the second s-roll-up operator, measures are aggregated to a specified drainage area ("Nibe-Bredning") at the DrainageArea level. We select all the possible topological cases where a parish intersects or within the drainage area, which means measures from the farms that are outside Nibe-Bredning are also aggregated to the level of this drainage area. In order to prevent this, the query needs to include an s-drill-down operator (Remark 13) to farms from Parish level and apply a spatial Boolean predicate to select the farms *within* the drainage area and then aggregate.

```
# 1 - s-roll-up #
SELECT ?parish (SUM(?animalCount) AS ?totalAnimals)
WHERE { ?obs rdf:type qb:Observation ;
   gfs:numberOfAnimals ?animalCount;
   gfs:farmID ?farm ;
   gfs:parishID ?parish .
?farm gfs:farmLocation ?farmGeo .
?parish gfs:parishCenter ?parishCent .
# Inner select for finding the
# closest farms to the parish centers #
   {SELECT ?farm1 (MIN(?distance) AS
   ?minDistance) WHERE
   { ?obs rdf:type ab:Observation ;
      gfs:farmID ?farm1;
      gfs:parishID ?parish1 .
   ?farm1 gfs:farmLocation ?farm1Geo .
   ?parish1 gfs:parishCenter ?parish1Cent.
   BIND (bif:st_distance (?farm1Geo, parish1Cent)
   AS ?distance) } GROUP BY ?farm1 }
   FILTER (?farm = ?farm1 && bif:st_distance
   (?farmGeo, ?parishCent) = ?minDistance )}
GROUP BY ?parish
```

```
# 2 - s-roll-up #
SELECT ?drainageArea (AVG(?nitRed) AS ?avgNitRed)
WHERE { ?obs rdf:type qb:Observation ;
   gfs:location ?farmLocation ;
   gfs:nitrogenReduction ?nitRed ;
   gfs:parishID ?parish.
   ?parish qb4o:memberOf gfs:parish ;
      gfs:parishPolygon ?parishGeo ;
      skos:broader ?drainageArea .
   ?drainageArea gfs:memberOf gfs:drainageArea ;
      gfs:waterPolygon ?drainageGeo ;
      gfs:waterName ?drainageName .
FILTER (bif:st_within(?parishGeo, ?drainageGeo)
|| bif:st_intersects(?parishGeo, ?drainageGeo)
&& ?drainageName = "Nibe-Bredning")}
GROUP BY ?drainageArea
```

Remark 13. (S-Drill-down) The s-drill-down operator disaggregates measures of a given cube by using an aggregate function and a spatial function along a spatial dimension's hierarchy. It is the inverse operator of s-roll-up, therefore s-drill-down disaggregates the previously summarized data to a child level in order to obtain measures at a finer granularity.

6.2 Nested SOLAP Operations

A nested set of SOLAP operators can be designed with the pattern $(s\text{–}dice_2(s\text{–}roll\text{–}up_1(\ldots s\text{–}roll\text{–}up_k(s\text{–}slice_1(\ldots s\text{–}slice_n(s\text{–}dice_1(DataCube)))))))$. Initially a sub-cube is selected from the (spatial) data cube with the first s-dice. Afterwards, a number of s-slices can be applied, which is followed by a series of s-roll-ups. Finally, the expression ends with another s-dice for getting the final sub-cube at a coarser granularity by filtering the aggregated measures. In the following, we present a nested SOLAP operation example for the running case GeoFarmHerdState spatial data cube.

Example 13 ($(^3s\text{–}roll\text{–}up(^2s\text{–}slice(^1s\text{–}dice(GeoFarmHerdState))))$) This pattern represents a typical nested SOLAP operation that can be paraphrased for the running use case as follows: [1]Filter the farm states located within a 2 km distance from the center of their parish and [2]slice on the parish which has the most number of topological relations (intersects, within) with a drainage area, [3]average the nitrogen reduction potential of the drainage areas intersecting with the parish.

7 Discussion and Perspectives

In the following, we give and evaluate the steps of our process with respect to the guidelines for publishing governmental linked data [18]. We discuss the particular challenges that we encountered and possible future improvements.

1) Specification. The first step is to specify the scope of the data by identifying and analyzing the data sources. We identified the data sources for the domains of CHR data, Environmental data, Geographical data, and CVR data as described in Section 4. In order to find the correct relations between these domains we had to search documentations (i.e., [17] and [16]) and acquire knowledge about the domains' interests. As the purpose is to publish open data, the definition of an Open Data license is also required at this level.
2) Modeling. We used the spatially extended MultiDim model [19] for designing the MD conceptual schema of the use case spatial data cube (Fig. E.2) from the collected flat data sets. This process requires good knowledge of spatial data warehouses and its concepts. In order to model the spatial data cube in RDF, QB4SOLAP provides the state-of-the-art semantic spatial data cubes. Therefore, we annotate the designed use case conceptual schema with QB4SOLAP.

Modeling the RDF data with QB4SOLAP provides all the core concepts of spatial data warehouses (i.e., spatial dimensions, spatial levels, and spatial hierarchies) for spatial data on the SW. Therefore, QB4SOLAP conveniently handles the conceptual modeling process of DWs on the SW and clearly describes the certain relations that should be considered during the logical modeling process (e.g., cardinality and topological relationships to create integrity constraints for ER models).
3) Generation. This step of the overall process involves the most complex tasks. In order to fully generate a spatial data cube in RDF the following sub-processes are performed: transformation and data conciliation.

3.1) Transformation. The RDF triples were generated with ad-hoc C# code for mapping from relational CSV files to RDF. In total, 12 CSV files were organized based on the relational representation (snowflake schema) of an MD conceptual model such that: We obtained *one* fact table with foreign keys of the related dimension (base) levels and measures, *four* tables with each dimensions' base level, and *seven* tables for the remaining levels along the hierarchies. These 12 tables are related by referential integrity constraints. Every level table also records the level attributes and attribute values. In order to create this relational CSV files, we pursued a number of data conciliation activities as described in the following item.

3.2) Data Conciliation. Initial data sets are downloaded in different formats i.e., SHP format for CHR, Environmental, and Geographical data; CSV

format for CVR data. In order to create the desired relational implementation of the use case spatial data cube, we used the unique identifiers (i.e., CVR and CHR numbers) or utilized spatial joins by joining attributes from one geometry feature to another based on the spatial containment relationship. For instance, we overlaid the point coordinates of the farms from CHR data and polygon coordinates of three environmental data sets in order to intersect and find the soil quality measurements for NitrogenReduction, NitrateClass and PhosphorClass of each farm. Another interesting spatial join is utilized for relating the Parish level members with DrainageArea level members. Since there is an $(n-n)$ cardinality relationship, some parishes intersect with more than one drainage area, thus we used topological relationships (intersects and within) to find the related child and parent level members. For interacting and handling the spatial data, we used QGIS with integration to PostGIS[12]. We used PostgreSQL to create and export relational tables.

The lack of tools for mapping a spatial multidimensional model to the relational model has been an impediment since we have to use topological relationships, where there is an $(n-n)$ cardinality relationship. Therefore, semantic ETL for data warehouses [20] is an important research topic, which requires improvements also for spatial data. Semi-automated tool support of geo-semantic ETL for publishing data warehouses on the SW is a promising improvement for handling the above processes such that the spatial joins can be processed efficiently. Before publishing the final RDF data, a comprehensive data cleansing step is essential for removing redundant columns and cleaning the noise due to unescaped characters, denormalized spatial literals, and encoding problems.

4) Publication. In order to store and publish the RDF data we chose the Virtuoso Universal Server as a triple store. The details about the SPARQL endpoint can be found on the project page `http://extbi.cs.aau.dk/GeoFarmHerdState`.

Publication of metadata in Danish and English languages should be completed. Also for enabling efficient discovery of published spatial data cubes, adding an entry of the data in the CKAN repository (`datahub.io`) is required.

5) Exploitation. The goal of our research is to re-use open government data and publish it as spatial data cubes on the SW for advanced multidimensional analysis. Therefore, we show how to query in SPARQL with SOLAP operators.

We recognize the need for non-expert SW users to write their spatial analytical queries in our high-level SOLAP language instead of the lower-level complex SPARQL language. Thus, a query system with a GUI that can interpret spatial data cube schemas for allowing users to perform high level SOLAP operations is ongoing work. Performing SOLAP queries in SPARQL to work over multiple RDF cubes with *s-drill-across* and supporting spatial ag-

[12]QGIS: `http://www.qgis.org/` PostGIS: `http://postgis.net/`

gregation (*s-aggregation*) over spatial measures are other important improvements on exploitation of spatial data cubes.

8 Conclusion and Future Work

The need for spatial analytical queries on the Semantic Web increases constantly with regularly published open government data, but there is a lack of effective solutions and efficient models. As a first attempt to publish spatial data cubes from open data, we have shown that the QB4SOLAP vocabulary can be used to link Danish government data that is published in different domains. First, we have studied the use case data sets thoroughly with corresponding regulations and requirements in order to satisfy cross-domain interests (e.g., tracking soil quality in livestock farms and farm animals density on drainage areas etc.). Second, we have conciliated the flat data sets in order to model the MD concepts of a spatial data cube. Third, we described the most popular individual SOLAP operators and a nested SOLAP operation pattern with examples and their SPARQL implementation.

In this paper, the QB4SOLAP vocabulary is validated by a non-trivial spatial use case. As a proof of concept, we showed that linking spatial (governmental open) data on the Semantic Web can be achieved at an advanced level, not solely linking spatial open data on the SW but also modeling this data for advanced analytical queries with SOLAP operations.

Several directions are interesting for future research: developing a geo-semantic ETL tool to support the process of creating spatial data cubes on the SW, a GUI for non-expert users to perform SOLAP operations on SW spatial cubes, extending the use case and implementing advanced SOLAP queries, such that; as s-drill-across and s-aggregation on the SW.

References

[1] J. B. Arendt, "Denmark releases its digital raw material," http://uk.fm.dk/news/, Ministry of Finance of Denmark, Denmark Ministry of Finance, October 2012.

[2] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Semantic Technology: 4th Joint International Semantic Technology Conference (JIST'14)*, vol. 8943. Springer, 2014, pp. 178–186, https://dx.doi.org/10.1007/978-3-319-15615-6_13.

[3] N. Gür, K. Hose, E. Zimányi, and T. B. Pedersen, "Modeling and Querying Spatial Data Warehouses on the Semantic Web," in *Semantic Technology: 5th Joint International Semantic Technology Conference (JIST'15)*,

vol. 9544. Springer, 2015, pp. 1–20, https://dx.doi.org/10.1007/978-3-319-31676-5_1.

[4] N. Gür, T. B. Pedersen, E. Zimányi, and K. Hose, "A Foundation for Spatial Data Warehouses on the Semantic Web," *Semantic Web Journal*, vol. 9, no. 5, pp. 557–587, 2018.

[5] A. Abelló, O. Romero, T. Pedersen, R. Berlanga Llavori, V. Nebot, M. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, no. 2, pp. 571–588, 2014, https://doi.org/10.1109/TKDE.2014.2330822.

[6] D. Pedersen, K. Riis, and T. B. Pedersen, "Query optimization for OLAP-XML federations," in *Proceedings of the 5th International Workshop on Data Warehousing and OLAP (DOLAP'02)*. ACM, 2002, pp. 57–64.

[7] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *The Semantic Web: ESWC 2012 Satellite Events*, vol. 7540. Springer, 2012, pp. 87–101, https://dx.doi.org/10.1007/978-3-662-46641-4_7.

[8] R. Cyganiak, D. Reynolds, and J. Tennison, "The RDF Data Cube Vocabulary," 2014.

[9] B. Kämpgen and A. Harth, "OLAP4LD–A framework for building analysis applications over governmental statistics," in *The Semantic Web: ESWC'14*. Springer, 2014, pp. 389–394.

[10] W3C, "Data Cube Implementations," 2014, https://www.w3.org/2011/gld/wiki/Data_Cube_Implementations.

[11] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web using QB4OLAP," in *Data Warehousing and Knowledge Discovery (DaWaK'14)*, vol. 8646. Springer, 2014, pp. 45–56, https://dx.doi.org/10.1007/978-3-319-10160-6_5.

[12] A. Matei, K.-M. Chao, and N. Godwin, "OLAP for Multidimensional Semantic Web Databases," *BIRTE*, pp. 81–96, 2015.

[13] P. Zhao, X. Li, D. Xin, and J. Han, "Graph Cube: On Warehousing and OLAP Multidimensional Networks," *SIGMOD*, pp. 853–864, 2011.

[14] K. A. Jakobsen, A. B. Andersen, K. Hose, and T. B. Pedersen, "Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries," *COLD*, 2015.

[15] M. Perry and J. Herring, "GeoSPARQL: A Geographic Query Language for RDF Data," *OGC Implementation Standard*, 2012.

[16] Danish Ministry of the Environment, "Consolidated Act on Livestock Farming Environmental Approvals," 2012, http://eng.mst.dk/media.

[17] Nitrates Directive, "Danish nitrate action programme 2008-2015 regarding the nitrates directive; 91/676/eec," http://eng.mst.dk/media/mst/Attachments/DanishNitrateActionProgramme2008201507092012.pdf, Nitrates Directive, Tech. Rep., 2012.

[18] B. Villazón-Terrazas, L. Vilches-Blázquez, O. Corcho, and A. Gómez-Pérez, "Methodological Guidelines for Publishing Government Linked Data," in *Linking Government Data*. Springer New York, 2011, pp. 27–49. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1767-5_2

[19] A. Vaisman and E. Zimányi, "Spatial data warehouses," in *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[20] R. P. Deb Nath, K. Hose, and T. B. Pedersen, "Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses," in *Proceedings of the 18th International Workshop on Data Warehousing and OLAP (DOLAP'15)*. ACM, 2015, pp. 15–24, https://doi.org/10.1145/2811222.2811229.

Paper F

Multidimensional Enrichment of Spatial RDF Data for SOLAP

Nurefşan Gür, Torben Bach Pedersen, Katja Hose, and Mikael Midtgaard

The paper is under preparation for submission to
Semantic Web Journal

Abstract

Large volumes of spatial data and multidimensional (MD) data are being published on the Semantic Web (SW) has lead to new opportunities for advanced analysis such as spatial Online Analytical Processing (SOLAP). The RDF Data Cube (QB) and QB4OLAP Vocabularies have been widely used for annotating and publishing statistical and MD RDF data. Even though such statistical data sets might have spatial information (i.e., coordinates), lack of spatial semantics and spatial MD concepts in QB4OLAP and QB does not allow users to employ SOLAP queries over spatial data using SPARQL or a third-party SPARQL query generator tool - GeoSemOLAP *for SOLAP operators. QB4SOLAP Vocabulary fully supports annotating spatial and MD data on the SW, which allows users to query endpoints with GeoSemOLAP or SOLAP operators in SPARQL. In order to enable SOLAP on existing QB and QB4OLAP data on the SW, we propose a RDF2SOLAP enrichment module that can automatically annotate spatial MD concepts in QB4SOLAP. We present a wide range of enrichment algorithms by applying them on a non-trivial real world use case from governmental open data sets with complex geometry types.*

1 Introduction

Data warehouses (DWs), Online Analytical Processing (OLAP) tools and queries are well-established for interactive data analyses. DWs have multidimensional (MD) models and store large volumes of data. MD models locate data in an n-dimensional space, which are usually called *data cubes*. The *cells* of the cube represent the topic of analysis, and associate observation *facts* with (numerical) *measures* that can be aggregated. Spatial data cubes can also contain *spatial* measures, which can be aggregated with spatial functions. For example, a fact cube for *farms* has a numerical measure 'number of animals' in the farm and has 'farm's coordinates' as spatial measure. Facts are linked to *dimensions*, which provide contextual information, e.g., farm production, farm location, and farm livestock. Dimensions are organized into *hierarchies* with *levels*, e.g., parish of the farm, herd type of livestock that allow users to analyze and aggregate measures at different levels of detail. Levels have a set of *attributes* that describe the characteristics of the level members.

In traditional DWs, the location dimension is generally used as a conventional (non-spatial) dimension with alphanumeric data and thus given only with a nominal reference to places and areas, e.g., parish name. This does not allow applying spatial operations over the true spatial location data or deriving topological relations among the hierarchy levels of the location dimension, which are essential for enabling spatial OLAP (SOLAP) analysis. By including the geometric information of location data in MD models, we can significantly improve the analysis process (e.g., proximity analysis of locations) with additional perspectives by revealing dynamic spatial hierarchy levels and new spatial level members in SOLAP operations (details and examples in [1, 2]). Or by using the geometry attributes of level members, topological relations between the levels, and levels and facts can be implicitly specified. These topological relations are essential in order to correctly aggregate measures between the levels with many-to-many (N:M) cardinality relations.

The Semantic Web (SW) has evolved, from prominently focusing on data publishing to also supporting complex queries such as interactive analytical queries. Simultaneously, the data available on the SW has evolved from being simple, mostly alphanumerical data, to include complex data types such as geospatial data. There are many examples of governmental and statistical Linked Open Data (LOD) sets with geographical attributes. However, such datasets are not typically modeled with multidimensional (MD) concepts. Thus, they cannot be queried with interactive analytical queries - OLAP on the SW. Even though, several platforms and tools for Business Intelligence (BI) and data warehouses emerged in the recent years [3], there is still a lack of common standard to model and publish (geo) semantic cubes on the

SW [1].

Some of the statistical datasets on the SW, which have observations and measures (that are well suited for analytical queries) are published using the RDF Data Cube Vocabulary (QB) [4], the current W3C standard. However, QB lacks the underlying structural metadata for multidimensional models and OLAP operations (explained in details in Section 7). Well-defined structural metadata is required in order to translate the OLAP queries into the underlying technology, which is SPARQL 1.1 [5], since the focus of interest is MD data on the SW [2]. QB4ST [6] is a recent attempt to define extensions for spatio-temporal components to RDF Data Cube (QB). However, it has the inherent multi-dimensional modeling limitations of the QB vocabulary.

In order to address the MD modeling challenges of the QB vocabulary, QB4OLAP [7] has been proposed, which reuses QB definitions by adding the required MD schema semantics. There is already a significant number of data sets published using QB vocabulary. The QB2OLAP enrichment module [8] helps users to semi-automatically produce a QB4OLAP description of a QB data cube by adding the necessary MD semantics (e.g., the hierarchical structure of the dimensions) and the corresponding instances to populate the dimension levels. However, QB4OLAP annotation only covers *non-spatial* MD data cube concepts and its operations. Even though such statistical data sets have spatial information, not annotating the spatial MD concepts (e.g., spatial hierarchy levels such as administrative regions) hinders querying the data with interesting spatial OLAP operations.

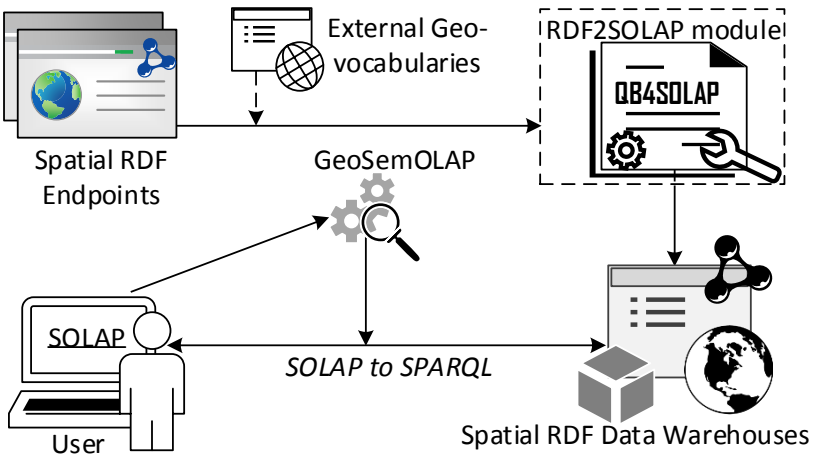


Fig. F.1: Future vision of SOLAP on the SW

Problem Definition. In the current state of the Semantic Web, spatial OLAP (SOLAP) queries are not possible through the existing spatial RDF endpoints. If a (spatial) data warehouse user would like to query spatial RDF data from the Semantic Web with SOLAP operations, the user needs to download the RDF data, map it to a relational data model (e.g., with a snowflake schema), and then import it into a traditional spatial data warehouse in order to query it with SOLAP queries, which is slow, labor-intensive, and stores the data in a non-open format.

There are existing tools and vocabularies for (spatial) data warehouse users to use on the Semantic Web. The QB4SOLAP vocabulary [9] allows spatial data warehouse users to publish their data on the Semantic Web with spatial multi-dimensional concepts. High-level SOLAP operators and how to translate them into SPARQL are defined with query generation algorithms [1]. Based on these algorithms, GeoSemOLAP [2] is developed to allow users querying with SOLAP operations on the Semantic Web without knowledge of SPARQL or RDF via a graphical user interface. However, GeoSemOLAP is restrained to RDF data sets, which are annotated with QB4SOLAP.

In order to minimize the user effort for querying existing spatial RDF endpoints (that are already published in other vocabularies, e.g., RDF Data Cube Vocabulary, QB4OLAP Vocabulary) with spatial analytical queries (SOLAP), an automated way of annotating spatial metadata with QB4SOLAP from the existing endpoints is necessary. Therefore, we propose an *RDF2SOLAP enrichment module* that can operate at the back-end of GeoSemOLAP.

Contributions. In order to address this issue our main contributions are:

* We show that the QB4SOLAP vocabulary yields the need for fully-fledged spatial data warehouse concepts, by demonstrating the running use case examples from real world governmental open data sets from various domains (i.e., environment, farming) with complex geometry types.

* A detailed explanation and comparison of RDF data examples, which are depicted as graphs, and annotated both in QB4OLAP and QB4SOLAP vocabularies, then identifying the required spatial MD metadata and concepts (e.g., spatial hierarchies and topological relations) for SOLAP analysis based on the given comparison.

* Hierarchical enrichment algorithms for (1) detecting topological relations at explicit hierarchy steps with direct links between the level members; and (2) discovering topological relations at implicit hierarchy steps (without direct links between the level members).

* Factual enrichment algorithms for both implicit and explicit fact-level relations between the fact and level members.

* An automated way of re-defining a fact schema after factual enrichment, and association of spatial aggregate functions with spatial measures.

* Evaluation of our approach in terms of accuracy by comparing the number of topological relations found in RDF2SOLAP framework against two different environments (RDBMS, and GIS tool), which can operate with spatial data types.

Paper organization. The remainder of this paper is organized as follows. Section 2 defines the preliminary concepts used throughout the paper with a running use case example. Section 3 presents the system architecture for the MD enrichment process. Section 4 defines the RDF2SOLAP enrichment algorithms with necessary helper functions and formalization of (spatial) RDF data. Section 5 presents the implementation details along with interesting examples and discusses the challenges and implemented solutions. Section 6 presents the qualitative and quantitative evaluation with comparison baselines. Finally, Section 7 discusses related work and Section 8 concludes the paper with an outlook to future work.

2 Preliminaries

In this section, we explain the preliminary concepts of spatial data warehouses and spatial OLAP (SOLAP) (Section 2.1) and how to deploy them on the Semantic Web (Section 2.2) using QB4SOLAP vocabulary.

2.1 Spatial Data Warehouses and SOLAP

Data cubes and spatially extended cube concepts Data warehouses (DW) are based on the multidimensional (MD) model, which models data in an n-dimensional space, and they are often referred to as data cubes. A cube *schema* defines the structure of a cube with MD concepts. The cells of the cube represent *(observation) facts* with a set of attributes called *measures*. Facts are linked to *dimensions*, which are the axes of an MD space and provide perspectives to analyze the data. Dimensions are organized into *hierarchies*, which allow users to aggregate measures at different granularities along the levels of a hierarchy. Hierarchies are composed of *levels*, which have a set of *attributes* that describe the characteristics of the level members. Each *level member* is defined by its attributes and attribute values.

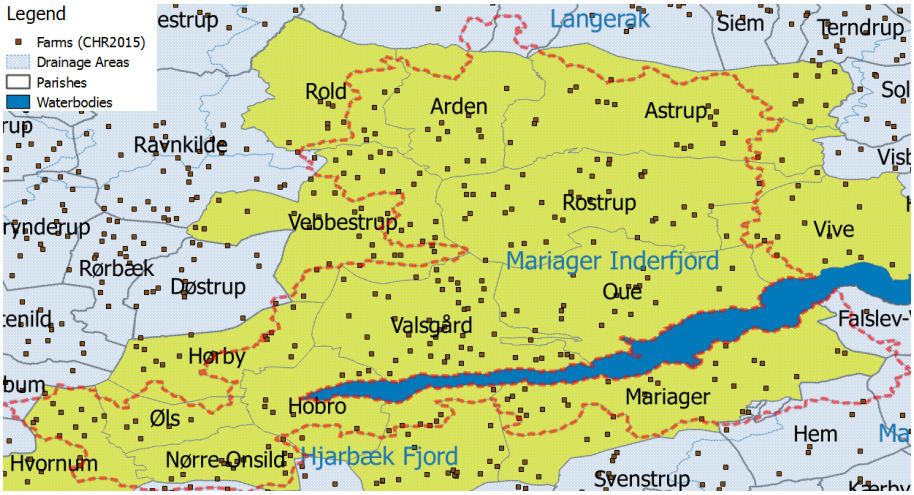


Fig. F.2: GeoFarmHerdState – Parish, Farm, and Drainage area instances

Cube members are MD concepts, which are defined at the *instance* level, and these are composed of level members, attributes of level members, *partial order* on level members, and fact members. A *hierarchy step* between levels (a child level and a parent level) defines a set of *roll-up* relations, where each relation relates a child level member to a parent level member. These roll-up relations define a partial order between level members with a *cardinality* relation. The cardinality (1:1, 1:N, N:1, N:M) describes the number of members in one level that can be related to a member in the other level for both child and parent levels.

Spatial data warehouses (SDW) extend a DW by storing geometries such as *point*, *line*, and *polygon* in the values of spatial measures and values of level attributes for spatial dimensions. The spatially extended MD schema of an SDW has spatial dimensions, spatial hierarchies, spatial levels [10], spatial hierarchy steps, and topological relations[1] (in addition to cardinality relations) between spatial levels for each spatial hierarchy step [9]. Similar to conventional DWs, facts of an SDW can be associated with numeric measures, which are using aggregation functions such as `SUM`, `AVG`, etc. A fully extended spatial MD schema of an SDW should also define spatial measures, which have geometries and spatial aggregate functions such as `UNION`, `CONVEX HULL`, etc., on those spatial measures. For a detailed explanation of SDW concepts we refer the reader to [12].

[1]Topological relations are Boolean spatial predicates that specify how two spatial objects are related to each other, e.g., *within*, *intersects*, *touches*, *crosses* and etc. [11].

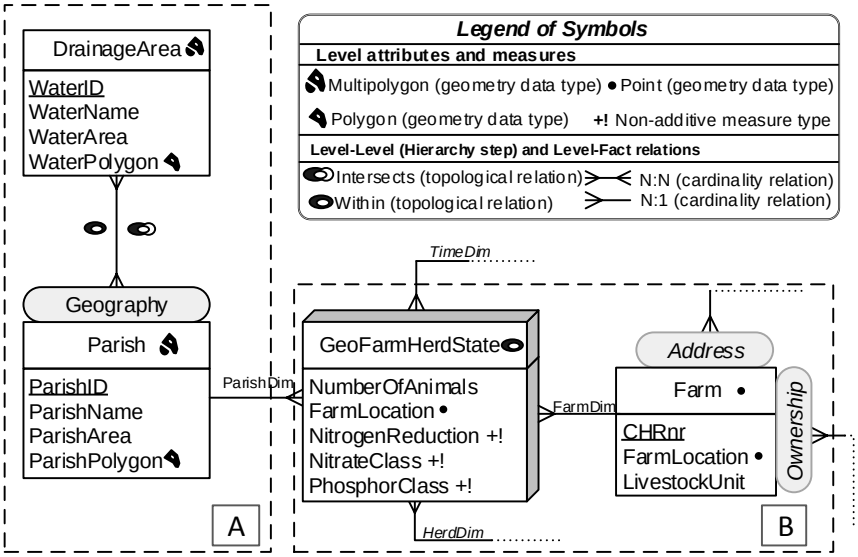


Fig. F.3: GeoFarmHerdState – Conceptual MD schema of livestock holdings data (Spatial concepts)

OLAP and spatial OLAP operations DWs are commonly used to store large volumes of data for decision support with On-Line Analytical Processing (OLAP) operations. Spatial OLAP (SOLAP) integrates the features of OLAP tools and geographical information systems (GIS) [13]. SOLAP enables advanced analytical processing by taking the spatial information in the cube into account.

For example, a spatial data cube of livestock holdings in farms (referred to as *GeoFarmHerdState* in the rest of this paper) defines the farm location as a spatial measure, which is linked to the observation facts. In order to derive perspectives and relations on the state of the farms' livestock holdings (herds), spatial levels are defined: parishes and drainage areas. A sample set of the corresponding spatial data cube members are given in Figure F.2. The spatial MD concepts of the data cube are defined in the conceptual schema in Figure F.3, which depicts a simplified version of the GeoFarmHerdState spatial data cube without its non-spatial dimensions (full version of the GeoFarmHerdState cube is described in our previous work [14]). The cube has two spatial dimensions: *FarmDim* and *ParishDim*. The latter has a spatial hierarchy (*Geography*) with two spatial levels: *Parish* and *DrainageArea*. *FarmDim* on the other hand does not have a spatial hierarchy, despite its spatial (base) level: *Farm*.

The GeoFarmHerdState cube has spatial fact members of state of the farms within a time frame with different kind of measures, i.e., numeric

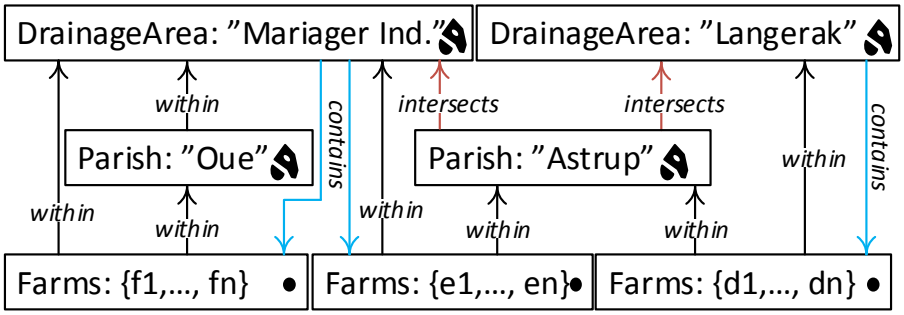


Fig. F.4: Hierarchy example for SOLAP

measures: *NumberofAnimals* in the farm and *NitrogenReduction* potential of the farm land/soil, spatial measures: *FarmLocation* (Figure F.3)[2].

To evaluate SOLAP operations, spatial levels such as *Parish* and *DrainageArea* are used to aggregate measures at different levels of detail. Due to the *polygon* geometry of the spatial level members, there are two different roll-up relations for the hierarchy step between the Parish and DrainageArea levels, where a parish can be completely contained *within* a drainage area or a parish and a drainage area can *intersect*.

For example, parish “Oue" is *within* drainage area “Mariager Inderfjord". Thus all the farms that are *within* “Oue" are also *within* “Mariager Inderfjord". Whereas, parish “Astrup" *intersects* with drainage areas “Mariager Inderfjord" and “Langerak". Therefore, some farms that are *within* “Astrup" are *within* “Mariager Inderfjord", while the rest of the farms are *within* “Langerak". Figure F.2 displays a sample set of Parish and DrainageArea level members.

The possible roll-up relations for the example above are depicted in Figure F.4 with black and red arrows, respectively representing the topological relations *within* and *intersects*. Blue arrows show the topological relation *contains*, which are drill-down (inverse operation of roll-up) relations from DrainageArea level to Farm level.

Topological relations between the levels and levels and facts can be implicitly specified through the geometry attributes of their instances (level members and fact members). The relations between spatial levels enable processing spatial roll-up and drill-down through range queries with spatial predicates [15]. In terms of cardinality, there is an N:M relationship between level members since a parish may intersect with more than one drainage area

[2]Non-additive measures are also numeric measures, which are given in percentages or classified in numbers, therefore they cannot be meaningfully summarized by all aggregate functions i.e., `SUM`. However, depending on the semantics, other aggregate functions can be associated with them, e.g., `AVG` NitrogenReduction potential, `MAX` NitrateClass.

and vice versa. This induces the problem of computing measures incorrectly when a roll-up operation goes through an N:M relationship, which actually is the case between the Parish level and the DrainageArea level. For example, we would like to aggregate the measure NumberOfAnimals, from Parish level to the DrainageArea level with a roll-up query. In such a roll-up query, we might falsely aggregate the number of animals in farms that are contained *within* the parish, but not contained *within* the drainage area, since the parish *intersects* with another drainage area. In order to refine such an analysis, SOLAP operations are required, where a (spatial) drill-down should be applied to the lowest granularity - from Parish level members to GeoFarmHerdState fact members, and then a spatial roll-up (with within predicate) can be applied from fact members (Farm instances) to DrainageArea level members. This would prevent falsely aggregating the number of animals from the farms that are (spatially) disjoint to the corresponding drainage area.

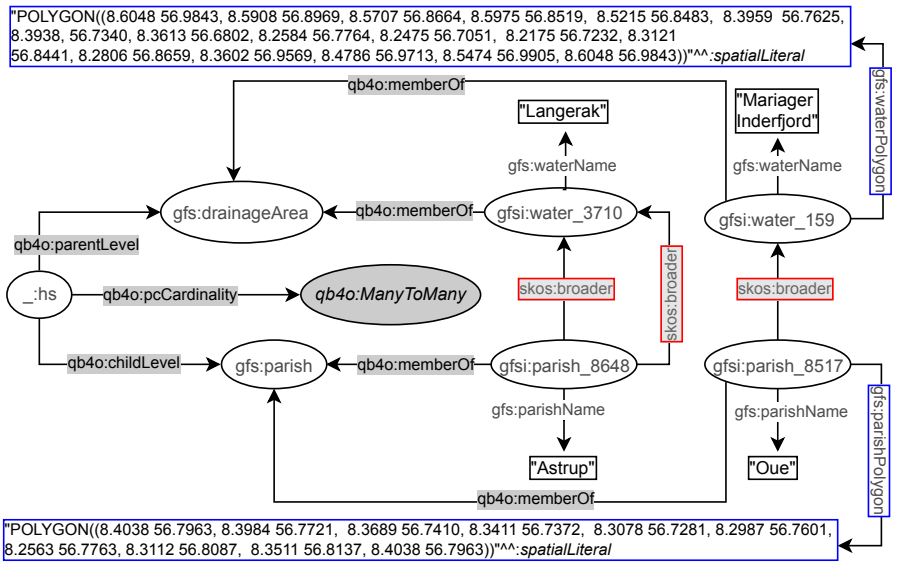


Fig. F.5: Hierarchy steps in QB4OLAP before multidimensional enrichment

2.2 QB4SOLAP: Spatial RDF Data Cube Vocabulary for SOLAP operations

There is an increasing amount of Linked Open Data (LOD) on the Semantic Web containing spatial information and numerical (statistical) data. This led to new opportunities for OLAP over spatial data using semantic web technologies and standards. Datasets on the SW use a standardized format: RDF

(resource description framework).

In order to enable SOLAP operations on the Semantic Web, a comprehensive RDF vocabulary is needed, i.e., annotation of the spatial hierarchy steps with topological relations. QB4SOLAP [1] is an RDF vocabulary, which allows MD data users to define *cube schemas* and *cube instances* as RDF triples. The QB4SOLAP vocabulary is an extension of QB4OLAP [7] capturing the semantics of spatial MD concepts (i.e., spatial hierarchy steps) that are essential for SOLAP operations. The QB4SOLAP Vocabulary is available on our project website[3]. The latest version is QB4SOLAP V1.3 and it is available via a persistent URL[4]. A comprehensive foundation on spatial data warehouses on the Semantic Web can be found in [1], which includes the detailed definitions with semantics of spatial MD concepts in RDF both at the schema level and instance level using QB4SOLAP.

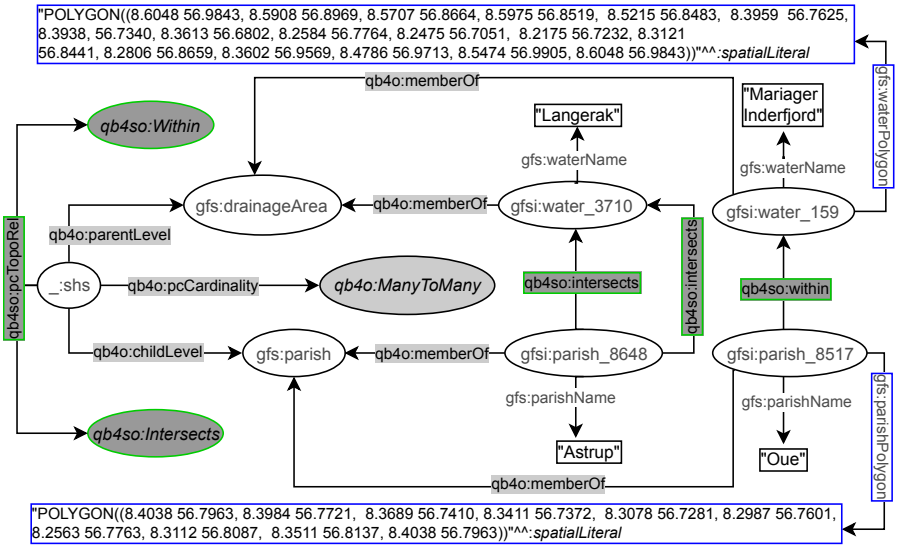


Fig. F.6: Spatial hierarchy steps in QB4SOLAP after multidimensional enrichment

In the following, we depict an example of a hierarchy step from `gfs:Parish` child level to `gfs:drainageArea` parent level (Figure F.5). In the figure, we prefix the schema elements (attributes, levels, etc.) of the (GeoFarmHerdState) cube with `gfs:` and instance data from the cube with `gfsi:`. The left-center part of Figure F.5 shows the hierarchy structure `_:hs`, between `gfs:parish` and `gfs:drainageArea` levels at the schema level with QB4OLAP vocabulary. QB4OLAP objects, classes and properties are prefixed with `qb4o:`. The levels (`gfs:parish` and `gfs:drainageArea`) are linked to the instances

[3] `https://extbi.cs.aau.dk/QB4SOLAP`

[4] `https://w3id.org/qb4solap#`

of level members (e.g., `gfsi:parish_8648`, `gfsi:water_3710` and etc.) by `qb4o:memberOf` property. The polygon geometry attributes are highlighted in blue boxes, on the top and the bottom of the figure. The coordinates recorded in the geometry attributes can be used to derive the topological relation between the level members by applying spatial boolean predicates (e.g., *instersects?*, *within?*) on the *polygon* geometries of the parish and drainage area level members.

However, QB4OLAP does not support annotating the topological relations that might exist between the level members at a hierarchy step. QB4OLAP uses only `skos:broader` property from SKOS (Simple Knowledge Organization System) [16] semantic relations for capturing the roll-up relations at hierarchy steps. The roll-up relations with `skos:broader` property are highlighted in red boxes in Figure F.5. The `skos:broader` property does not describe the nature of the roll-up relation with topological relations for spatial hierarchies. Therefore, QB4OLAP cannot capture the topological relations in a hierarchy step from Parish level to DrainageArea level or between these levels' members.

On the other hand, QB4SOLAP can define topological relations both at the schema level and the instance level. In Figure F.6, we prefix QB4SOLAP objects, classes and properties with `qb4so:` and highligh them in green lines. The left-center part of the figure shows the spatial hierarchy structure `:_shs`, which has a QB4SOLAP property `qb4so:pcTopoRel`, and that has two QB4SOLAP class instances `qb4so:Within` and `qb4so:Intersects`. This means, when we compare the geometry attributes of parish level members and drainage area level members, we discover two different topological relations (*within* and *intersects*) for all the (spatial) hierarchy steps between the parish and drainage area levels. And these relations are annotated at the schema level on the left-center part of Figure F.6.

Similarly, gfs:parish and `gfs:drainageArea` levels are linked to the instances of level members (e.g., `gfsi:parish_8648`) by `qb4o:memberOf` property. The explicit topological relations between the each level members along a spatial hierarchy step are depicted in the figure with `qb4so:intersects` or `qb4so:within` predicates, which are highlighted in green boxes (e.g., `gfsi:parish_8648` *intersects* with `gfsi:water_159` and `gfsi:water_3170` etc.).

In conclusion, QB4SOLAP can enable SOLAP operations by defining the semantics of spatial MD concepts both at the schema level and instance level. These semantics are essential for SOLAP operations and they are defined as extensions to the QB4OLAP vocabulary.

3 System Architecture

The importance of SOLAP to get accurate results in operations over spatial data warehouses is explained in Section 2.1. However, the RDF data cubes (with spatial attributes) on the Semantic Web are not always annotated with vocabularies that allow users to formulate SOLAP queries. In this section we present an overview of the MD enrichment flow from RDF QB to QB4OLAP data cubes and QB4OLAP to QB4SOLAP data cubes. Thus, the users can query the RDF data cubes with SOLAP queries.

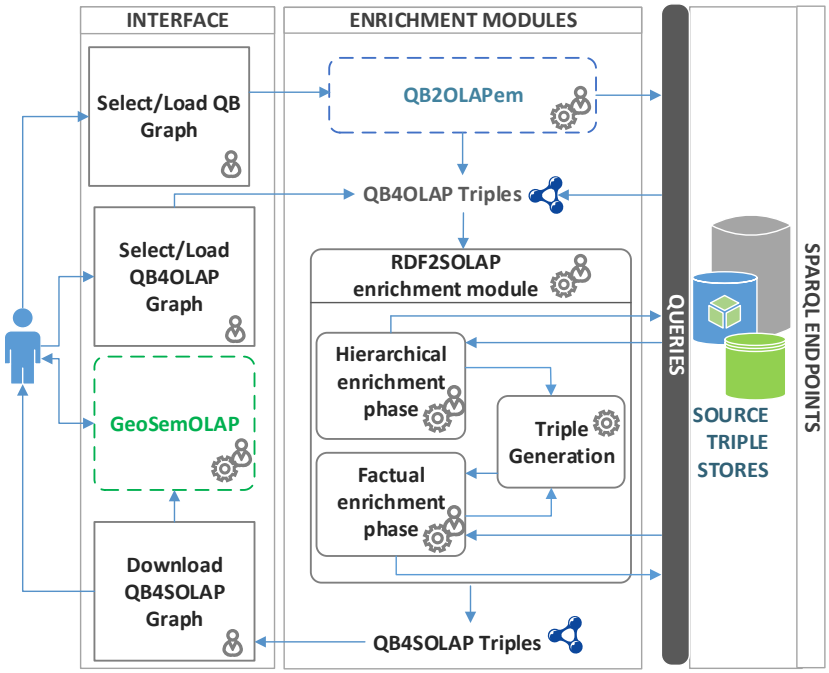


Fig. F.7: Multidimensional Enrichment Process

Multidimensional enrichment process flow is illustrated in Figure F.7 with three main architectural layers: Interface, Enrichment Modules, and SPARQL Endpoints. The first layer facilitates user interaction with the enrichment modules (i.e., QB2OLAPem) and third party tools (i.e., GeoSemOLAP). Our main contribution in this paper is the RDF2SOLAP enrichment module, which is the core of the second layer. The RDF2SOLAP enrichment module operates on QB4OLAP triples that either already exist in the original data or have been generated by QB2OLAPem enrichment module [8]. QB2OLAPem allows users to enrich an RDF QB dataset with QB4OLAP concepts and returns a graph of QB4OLAP triples.

The internal process flow of RDF2SOLAP enrichment module consists of three phases: Hierarchical enrichment phase, factual enrichment phase, and triple generation (phase). Hierarchical and factual enrichment phases iteratively perform the enrichment algorithms explained in Section 4. Both of these enrichment phases allow interaction with external SPARQL endpoints to enhance the enrichment process via potential spatial and multidimensional concepts that could be retrieved externally. The third phase is the triple generation, which creates QB4SOLAP triples that can be used in third party tools such as GeoSemOLAP. GeoSemOLAP allows users without knowledge of RDF and SPARQL to query with SOLAP operations by interactively formulating the queries using a GUI with interactive maps [2].

The third layer (SPARQL endpoints) allows both user-SPARQL endpoint interaction for retrieving QB or QB4OLAP graphs and system-SPARQL endpoint interaction where RDF2SOLAP enrichment module queries the external triple stores for enhancing the hierarchical enrichment and factual enrichment.

RDF2SOLAP is implemented in Javascript on Node.js platform with N3.js library for parsing the RDF triples in Javascript and Turfjs library for spatial analysis[5].

4 RDF2SOLAP Enrichment Algorithms

In this section, the algorithms in RDF2SOLAP enrichment module are presented. Our MD enrichment approach builds upon QB4OLAP triples that either already exist in the original data or have been generated by QB2OLAPem enrichment module [8] as depicted in Figure F.7. QB4OLAP defines only the non-spatial multidimensional semantics of RDF data, whereas QB4SOLAP enriches the MD semantics of RDF data with spatial concepts. The high-level formalizations of spatial and non-spatial multidimensional RDF data is defined in [1]. We use these formalizations to refer in the algorithms as input and output variables. In the following, we briefly explain the basic notations in RDF terms that are necessary for the algorithms.

The basic construct of RDF is a triple $t = (s, p, o)$ consisting of three components; s is the subject, p is the predicate, and o is the object. RDF triples are defined over $\mathcal{T} = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, where $\mathcal{I}$ is the set of *IRIs* (Internationalized Resource Identifiers), $\mathcal{B}$ is the set of *blank nodes*, and $\mathcal{L}$ is the set of *literals*. An object value can be a *literal* (i.e., string, spatial literal[6], integer etc.). Subjects and objects can be represented by a *blank node* for anonymous resources. Predicates are always represented by IRIs. A set of RDF triples is referred to as an RDF *graph* $\mathcal{G}$. We use superscript notation

[5] **N3.js:** `https://github.com/rdfjs/N3.js` **Turfjs:** `http://turfjs.org/`

[6] Spatial literals are represented as $\mathcal{L}_s$.

to represent the type of the QB4OLAP graph either as schema graph $\mathcal{G}^S$ or instance graph $\mathcal{G}^I$. The instance graph has entities from a use-case dataset as a set of RDF triples. The schema graph describes the structure (schema) of the dataset recorded in the instance graph. We use subscript notation to represent the MD concepts in RDF terms as a graph as exemplified in the following: $\mathcal{G}^I_{A(lm)}$ is the RDF instance graph for attributes of level members, i.e., in the use case example this graph corresponds to the set of triples in Listing 2, Lines 3-6 or Lines 9-13 and Lines 17-22. $\mathcal{G}^S_{HS(h)}$ is the RDF schema graph for hierarchy steps, i.e., in the use case example this graph corresponds to the set of triples in Listing 1.

We define the function $id(x) : \mathcal{G} \rightarrow \mathcal{I}$, that given an MD element x returns its identifier $\mathcal{I}$ from the graph $\mathcal{G}$. Similarly, we use superscript notation to indicate the type of the identifier from the schema graph ($\mathcal{G}^S$) and instance graph ($\mathcal{G}^I$), e.g., $id^S(a)$ for a schema identifier of a level (`gfs:parish` in Listing 2, Line 2 or in Listing 1, Line 2) and $id^I(lm)$ for an instance identifier of a level member (`gfsi:parish_8648` in Listing 2, Line 1 or Line 8).

The main contributions to the MD enrichment process is provided by two enrichment phases; *hierarchical enrichment phase* and *factual enrichment phase* in RDF2SOLAP enrichment module. These phases are explained in Section 4.1 and Section 4.2 respectively, with the corresponding algorithms.

4.1 Hierarchical enrichment phase

The hierarchical enrichment phase is built around spatial levels and their level members forming the spatial hierarchy of a dimension. Thus, by identifying the spatial relations between the spatial levels and their level members, we can find the spatial hierarchy steps and the possible topological relations for these hierarchy steps.

Each spatial hierarchy corresponds to a path of roll-up relationships between the child level and parent level: each of these roll-up relationships corresponds to a *spatial hierarchy step* (Section 2.1). An example of a (spatial) hierarchy with QB4SOLAP is given in Listing 1. Line 4 extends the QB4OLAP schema definitions by enriching the hierarchy step with possibility to annotate the spatial hierarchy steps with topological relations (see Section **??** for details and Section 2.2 examples).

```
## Spatial hierarchies in QB4SOLAP with topological relations##
1 _:_shs rdf:type qb4o:HierarchyStep ; qb4o:inHierarchy gfs:geography ;
2      qb4o:childLevel gfs:parish ; qb4o:parentLevel gfs:drainageArea ;
3      qb4o:pcCardinality qb4o:ManyToMany ;
4      qb4so:pcTopoRel qb4so:Within , qb4so:Intersects .
```

Listing F.1: Spatial Hierarchy structure in QB4SOLAP

In Listing 2, we give the GeoFarmHerdState spatial level members from Parish and Drainage Area levels. Lines 1-7 (Listing 2) represents the QB4OLAP annotation of a child level member from Parish level before multidimensional enrichment (with `skos:broader`), which are depicted in Figure F.5. Lines 8-14 represents the QB4SOLAP annotation of the same Parish level member after the multidimensional enrichment with topological relations, which are depicted in Figure F.6. Lines 15-22 represents the annotation of a parent level member from Drainage area level, which remains the same before and after multidimensional enrichment, since the hierarchy steps are defined with bottom-up relationships from child level to parent level and the roll-up relations and thus also the topological relations are annotated at the child level members of the hierarchy step.

```
   ## Parish (child) Level member before hierarchical enrichment##
1  gfsi:parish_8648 rdf:type qb4o:LevelMember ;
2       qb4o:memberOf gfs:parish ;
3       gfs:parishID 8648 ; gfs:parishName "Astrup" ;
4       gfs:parishArea 47,969 ; gfs:parishPolygon "POLYGON((8.438 56.796,
5       8.3984 56.7721, 8.3689 56.7410, 8.3411 56.7372, 8.3078 56.7281,
6       8.3112 56.8087, 8.3511 56.8137, 8.438 56.796))"^^geo:spatialLiteral ;
7       skos:broader gfsi:water_3710 , gfsi:water_159 .
   ## Parish (child) Level member after hierarchical enrichment##
8  gfsi:parish_8648 rdf:type qb4o:LevelMember ;
9       qb4o:memberOf gfs:parish ;
10      gfs:parishID 8648 ; gfs:parishName "Astrup" ;
11      gfs:parishArea 47,969 ; gfs:parishPolygon "POLYGON((8.438 56.796,
12      8.3984 56.7721, 8.3689 56.7410, 8.3411 56.7372, 8.3078 56.7281,
13      8.3112 56.8087, 8.3511 56.8137, 8.438 56.796))"^^geo:spatialLiteral ;
14      qb4so:intersects gfsi:water_3710 , gfsi:water_159 .
   ## DrainageArea (parent) Level member##
15 gfsi:water_159 rdf:type qb4o:LevelMember ;
16      qb4o:memberOf gfs:drainageArea ;
17      gfs:waterName "Mariager Inderfjord" ; gfs:waterArea 267,477 ;
18      gfs:waterPolygon "POLYGON((8.6048 56.9843, 8.5908 56.8969,
19      8.5707 56.8664, 8.5975 56.8519, 8.5215 56.8483,
20      8.3959 56.7625, 8.3938, 56.7340, 8.3613 56.6802,
21      8.2584 56.7764, 8.2475 56.7051, 8.2175 56.7232,
22      8.5474 56.9905, 8.6048 56.9843))"^^geo:spatialLiteral .
```

Listing F.2: GeoFarmHerdState level members, attributes, and *spatial* roll-up relations

We exploit QB4OLAP semantics such as *non-spatial* hierarchy steps and levels as a starting point to find the *spatial* hierarchy steps. We distinguish two cases:
Case 1: Finding *explicit* spatial hierarchy steps for QB4OLAP levels with `skos:broader` roll-up relations between their child-parent level members by

Algorithm 1: getSpatialValues($\mathcal{G}^I_{A(lm)}$) : $V_{s(a)}$

Input: $\mathcal{G}^I_{A(lm)}$
Output: $V_{s(a)}$
1 **begin**
2 $V_{s(a)} = \emptyset$; /*initialize output set as empty set*/
3 **foreach** $(id^I(lm)\ id^S(a_i)\ v_{a_i}) \in \mathcal{G}^I_{A(lm)}$ **do**
4 **if** v_{a_i} *is a* `geo:spatialLiteral` **then**
5 $V_{s(a)} \cup = \{v_{a_i}\}$;
6 **return** $V_{s(a)}$

detecting spatial hierarchy steps in Section 30. For this case we assume that level members have direct `skos:broader` relations as depicted in Figure F.5 and Listing 2, Line 7 with `skos:broader` property.
Case 2: Finding *implicit* spatial hierarchy steps from QB4OLAP levels without direct roll-up relations through the `skos:broader` property. In this case we assume that the level members are only defined by the `qb4o:memberOf` property as shown in Listing 2, Line 2 but *do not* have the `skos:broader` roll-up relation as given in Line 7. In this case, it is still possible to *discover spatial hierarchy steps* by finding the spatial (topological) relations between level members through their attributes as explained in Section 19.

Spatial helper functions

In order to address the cases explained above, we need two spatial helper functions; for retrieving spatial from a set of given attribute values (Algorithm 1, `getSpatialValues`), and for relating the spatial attributes (Algorithm 2, `relateSpatialValues`), which are explained as follows.

Algorithm 1 (`getSpatialValues`) **.** The first helper function gets an input graph of attributes of level members $\mathcal{G}^I_{A(lm)}$ and returns a set of spatial attribute values $V_{s(a)}$. For example, the function could receive Lines 3-6 from Listing 2 as an input. In the algorithm, Lines 3 and 4 checks the values v_{a_i} of each attribute $id^S(a_i)$ (e.g., `gfs:parishName`, `gfs:ParishArea`, etc.), if the value is a type of `geo:SpatialLiteral` (e.g., the `POLYGON` geometry value linked to the `gfs:parishPolygon` attribute), then the value is incrementally added to the output set $V_{s(a)}$[7] in Line 5.

[7]Note that a level member might have the polygon geometry type for the parish borders and have the point geometry type for the parish center, therefore a set of spatial values is required.

Table F.1: Topological relations for Hierarchy Steps (✓: hierarchically and topologically applicable, ×: topologically not applicable, –: hierarchically not applicable)

Roll-up Relations	child level	point (pt.)			line (ln.)			polygon (po.)		
	parent level	pt.	ln.	po.	pt.	ln.	po.	pt.	ln.	po.
Topological Relations	within	×	✓	✓	–	✓	✓	–	–	✓
	contains	–	–	–	–	–	–	–	–	–
	intersects	✓	✓	✓	–	✓	✓	–	–	✓
	touches	×	×	×	–	✓	✓	–	–	✓
	overlaps	×	×	×	–	✓	✓	–	–	✓
	crosses	×	×	×	–	✓	✓	–	–	×
	coveredBy	×	×	×	–	×	✓	–	–	✓
	covers	–	–	–	–	–	–	–	–	–
	equals	✓	×	×	–	✓	×	–	–	✓

Algorithm 2: (`relateSpatialValues`). The next helper function is designed based on Table F.1, w.r.t. the geometry values of the child-parent level members and based on the structure of a hierarchy step. We prepared Table F.1 with topological relations based on DE-9IM[8]. We consider only the three simple geometry types, *point, line, and polygon* as the spatial attribute values of child-parent level members in roll-up relations, excluding complex geometry types such as multi-polygon, multi-point etc. The possible topological relations that can occur in a spatial hierarchy step with a roll-up relation from child level to parent level are marked with check sign (✓) in the table. Topological relations such as *contains* and *covers* are not *hierarchically applicable* since a spatial child level member cannot contain or cover a spatial parent level member. For these relations, we mark the complete rows with minus sign (–) in the table, since they are not hierarchically applicable. Similarly, we mark the complete columns of *line-point*, *polygon-point*, and *polygon-line* roll-up relations with the minus sign (–), since these are also not hierarchically applicable. This is because, we assume that in the instance data, a parent level member should always have a spatial attribute of a geometry type of the same or higher dimensionality of its child level member (a point is 0-dimensional, a line is 1-dimensional and a polygon is 2-dimensional).

For example, a child level member with a spatial attribute of line geometry can only have parent level member(s) with spatial attributes of line or polygon geometries but not point geometry. We mark the *topologically not applicable* relations with cross sign (×) according to the DE-9IM model (e.g, a line cannot overlap a polygon).

In Figure F.8, we depict the hierarchically and topologically applicable topological relations from Table F.1. We simplified them by generalizing the

[8]DE-9IM (Dimensionally Extended Nine-Intersection Model) is a topological model that describes spatial relations of two geometries in two dimensions [11].

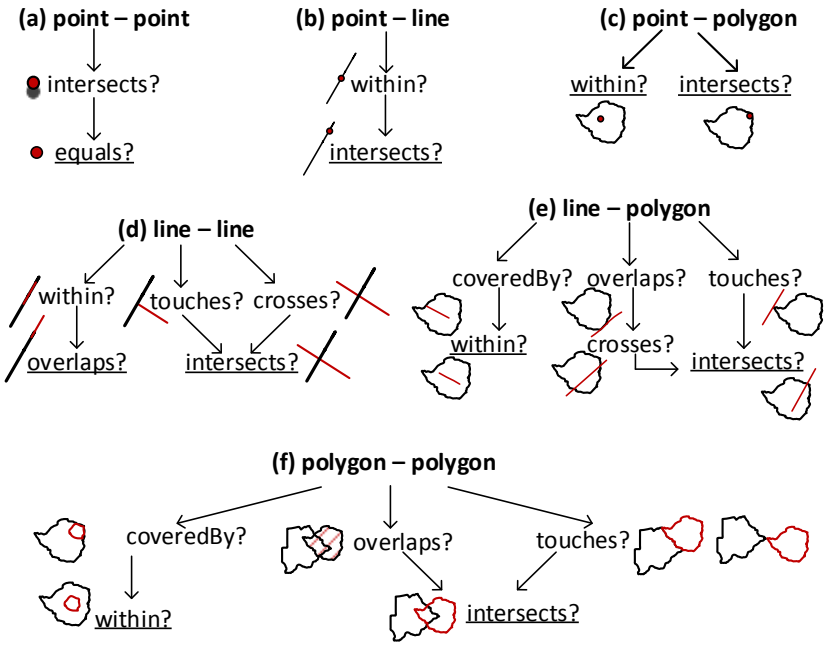


Fig. F.8: Simplifying Topological Relations

possible relations (e.g., if a line *touches* or *crosses* with another line at one point, they are both classified as *intersects* in Fig. F.8(d)). The most general relations are underlined in Fig. F.8 for each pair of geometry types (Fig. F.8(a), (b), (c), (d), (e), and (f)).

In Algorithm 2 `relateSpatialValues`, we only consider these general topological relations that have higher probability to satisfy the corresponding spatial predicates. For example, the topological relation *intersects* has the highest probability to satisfy from the DE-9IM matrix [11]. We generalize the similar spatial predicates to the ones that have higher probability to occur in a 2-dimensional space. For example, we generalize the relations such as a line *overlaps* (along the border of) a polygon can be generalized to the relation - a line *crosses* a polygon at a minimum two points, which can be later generalized to the relation - a line *intersects* a polygon at a (minimum) single point as in Figure F.8(e). Similarly, a line *touches* a polygon at a single point can be generalized to the relation - a line *intersects* a polygon at a (minimum) single point.

The topological relation *coveredBy* requires an area of a geometry, therefore it is applicable only in line-polygon and polygon-polygon relations (Figure F.8(e) and F.8(f)). For simplicity reasons, we choose to generalize them as

the *within* topological relation. In the algorithm we also prioritize to check the topological relations based on the compared geometry types. If the spatial attribute values to relate are *point* and *polygon* geometry types, as in Fig. F.8(c), it is more likely that a *point* is *within* a *polygon* than a *point intersects* a *polygon* in the instance data.

Therefore, we initially check for a more probable relation in the algorithm. For example, for point-polygon relations case in Algorithm 2, Line 10: initially, *within* spatial predicate is checked in the if statement (Line 11), then *intersects* spatial predicate is checked in the else if statement (Line 13). After checking all the possible combinations of the spatial attribute values in `switch case`, a topological relation is returned from the algorithm (Line 30).

Now, we have given the necessary spatial helper functions, we present the main algorithms in the following Sections 30 and 19 for finding the spatial hierarchy steps.

Detecting spatial hierarchy steps

In this section, we present the algorithm for *Case 1*, given in the beginning of Section 4.1, to find the *explicit* spatial hierarchy steps for QB4OLAP levels with `skos:broader` roll-up relations between their child-parent level members.

Algorithm 3 (`detectspatialHS`)**.** The input variables for Algorithm 3 are the instance graphs of attributes of level members $\mathcal{G}^I_{A(lm)}$ and roll-up relations of the hierarchy steps $\mathcal{G}^I_{RU(hs)}$ between the level members. The RDF graph formulation of the attributes of the level members $A(lm)$ is: $\mathcal{G}^I_{A(lm)} = \bigcup_{i=1}^{p}\{(id^I(lm) \;\; id^S(a_i) \;\; v_{a_i}) \mid lm \rightsquigarrow v_{a_i}\}$. Here, we denote by $lm \rightsquigarrow v_{a_i}$ that a level member lm has value v_{a_i} for attribute a_i (e.g., Listing 2, Lines 3-6, Lines 9-13 and Lines 17-22). The RDF graph formulation of the roll-up relations $RU(hs)$ is: $\mathcal{G}^I_{RU(hs)} = \bigcup_{i=1}^{k}\{(id^I(lm_c)$ `skos:broader` $id^I(lm_p)) \mid lm_{c_i} \sqsubseteq lm_{p_i}\}$. Here, we denote by $lm_{c_i} \sqsubseteq lm_{p_i}$ the partial order between level members, where a child level member lm_{c_i} rolls up to a parent level member lm_{p_i}[9] (e.g., Listing 2, Line 7).

The output of Algorithm 3 is the instance graph of roll-up relations for the *detected* spatial hierarchy steps $\mathcal{G}^I_{RU(shs)}$ (e.g., Listing 2, Line 14). In Line 2, initially the output graph is initialized as an empty set. Next, in Line 3 we create two temporary graphs: $\mathcal{G}^I_{A(lm_c)}$ and $\mathcal{G}^I_{A(lm_p)}$ as empty sets[10], to keep triple patterns separately in two graphs for attributes of child and parent level members. We also create two temporary sets: $V_{s(a_c)}$ and $V_{s(a_p)}$ for keeping

[9] We use subscript c and p to distinguish values for child and parent level members.

[10] Remark: a set of RDF triples is referred to as an RDF graph

Algorithm 2: relateSpatialValues(v_{a_c}, v_{a_p}):$\texttt{topoRel}_i$

Input: v_{a_c}, v_{a_p}
Output: $\texttt{topoRel}_i$

```
1  begin
2      topoRel_i = null; /*geoType(v_a) function returns the geometry
         type of a given attribute value*/
3      switch (geoType(v_ac), geoType(v_ap)) do
4          case (POINT, POINT) do
5              if equals?(v_ac, v_ap) then
6                  topoRel_i= qb4so:equals
7          case (POINT, LINE) do
8              if intersects?(v_ac, v_ap) then
9                  topoRel_i= qb4so:intersects
10         case (POINT, POLYGON) do
11             if within?(v_ac, v_ap) then
12                 topoRel_i= qb4so:within
13             else if intersects?(v_ac, v_ap) then
14                 topoRel_i= qb4so:intersects
15         case (LINE, LINE) do
16             if intersects?(v_ac, v_ap) then
17                 topoRel_i= qb4so:intersects
18             else if overlaps?(v_ac, v_ap) then
19                 topoRel_i= qb4so:overlaps
20         case (LINE, POLYGON) do
21             if within?(v_ac, v_ap) then
22                 topoRel_i= qb4so:within
23             else if intersects?(v_ac, v_ap) then
24                 topoRel_i= qb4so:intersects
25         case (POLYGON, POLYGON) do
26             if within?(v_ac, v_ap) then
27                 topoRel_i= qb4so:within
28             else if intersects?(v_ac, v_ap) then
29                 topoRel_i= qb4so:intersects
30     return topoRel_i
```

Algorithm 3: detectSpatialHS($\mathcal{G}^I_{RU(hs)}, \mathcal{G}^I_{A(lm)}$) : $\mathcal{G}^I_{RU(shs)}$

Input: $\mathcal{G}^I_{A(lm)}$, $\mathcal{G}^I_{RU(hs)}$
Output: $\mathcal{G}^I_{RU(shs)}$

1 **begin**
2 $\quad\mathcal{G}^I_{RU(shs)} = \emptyset$; /*initialize output graph as emptyset*/
3 $\quad\mathcal{G}^I_{A(lm_c)} = \emptyset$; $\mathcal{G}^I_{A(lm_p)} = \emptyset$; $V_{s(a_c)} = \emptyset$; $V_{s(a_p)} = \emptyset$; $\texttt{topoRel}_i = null$; /*temporary variable and sets*/
4 $\quad$**foreach** $((id^I(lm_c)\ id^S(a_c)\ v_{a_c}), (id^I(lm_p)\ id^S(a_p)\ v_{a_p})) \mid (id^I(lm_c)\ id^S(a_c)\ v_{a_c}), (id^I(lm_p)\ id^S(a_p)\ v_{a_p}) \in \mathcal{G}^I_{A(lm)} \wedge (id^I(lm_c)\ \texttt{skos:broader}\ id^I(lm_p)) \in \mathcal{G}^I_{RU(hs)} \wedge lm_c \rightsquigarrow v_{a_c} \wedge\ lm_p \rightsquigarrow v_{a_p} \wedge lm_c \sqsubseteq lm_p$ **do**
5 $\quad\quad\mathcal{G}^I_{A(lm_c)} = \{(id^I(lm_c)\ id^S(a_c)\ v_{a_c})\}$;
6 $\quad\quad V_{s(a_c)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_c)})$;
7 $\quad\quad$**if** $V_{s(a_c)} \neq \emptyset$ **then**
8 $\quad\quad\quad\mathcal{G}^I_{A(lm_p)} = \{(id^I(lm_p)\ id^S(a_p)\ v_{a_p})\}$;
9 $\quad\quad\quad V_{s(a_p)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_p)})$;
10 $\quad\quad\quad$**if** $V_{s(a_p)} \neq \emptyset$ **then**
11 $\quad\quad\quad\quad$**foreach** $(v_{a_c}, v_{a_p}) \in V_{s(a_c)} \times V_{s(a_p)}$ **do**
12 $\quad\quad\quad\quad\quad\texttt{topoRel}_i = \texttt{relateSpatialValues}(v_{a_c}, v_{a_p})$;
13 $\quad\quad\quad\quad\quad$**if** $\texttt{topoRel}_i \neq null$ **then**
14 $\quad\quad\quad\quad\quad\quad\mathcal{G}^I_{RU(shs)} \cup = \{(id^I(lm_c)\ \texttt{topoRel}_i\ id^I(lm_p))\}$;
15 $\quad$**return** $\mathcal{G}^I_{RU(shs)}$

the spatial attribute values from the child and parent level members, and initialize them as empty sets in Line 3. A set of spatial attribute values is defined over spatial literals $\mathcal{L}_s$ as $V_{s(a)} = \{v_{a_1}, \ldots, v_{a_i}, \ldots v_{a_n} \mid 1 \leq i \leq n \wedge v_{a_i} \in \mathcal{L}_s\}$.

In the `foreach` loop in Line 4, we go through the elements of the input graphs $\mathcal{G}^I_{A(lm)}$ and $\mathcal{G}^I_{RU(hs)}$ that are fulfilling a specific criteria, which is having an *explicit* `skos:broader` relation between child and parent level members.

In Line 5, while iterating through the `foreach` loop, we assign the set of triples of child level members and their attributes to the temporary graph $\mathcal{G}^I_{A(lm_c)}$. This temporary graph is given in Line 6 as an input to the helper function `getSpatialValues` (Algorithm 1), which finds the spatial attribute values from the given graph, and returns a set of spatial attribute values (i.e.,

$V_{s(a_c)}$) that are found in the input graph. The output of the helper function ($V_{s(a_c)}$) keeps the spatial attribute values of the child level member $id^I(lm_c)$.

Next in Line 7, if $V_{s(a_c)}$ is not empty and has some spatial values of $id^I(lm_c)$, then we populate the next temporary graph $\mathcal{G}^I_{A(lm_p)}$ with its parent level $id^I(lm_p)$ and attributes of the parent level in Line 8.

Similar to Line 6, Line 9 calls the helper function `getSpatialValues` with the input graph $\mathcal{G}^I_{A(lm_p)}$ and the output of the function is assigned to the temporary set $V_{s(a_p)}$. If this set is also not empty (Line 10), we go through the pairs of values (v_{a_c}, v_{a_p}) of the child-parent level members (Line 11), which are selected from the temporary graphs $\mathcal{G}^I_{A(lm_c)}$ and $\mathcal{G}^I_{A(lm_p)}$.

In this loop, we call the next helper function `relateSpatialValues` (Algorithm 2), where the input is the spatial value pairs. The output value of this function is the topological relation between the corresponding child and parent level members, and it is assigned to the initially created temporary variable $\texttt{topoRel}_i$ (Line 12). If this value is not null (checked in Line 13), `relateSpatialValues` function returns a topological relation (Line 12) that is satisfied as shown with a check-mark (✓) from Table F.1.

Finally, the output graph for *spatial* hierarchy steps $\mathcal{G}^I_{RU(shs)}$ is incrementally generated by adding the triple pattern with the topological relation (Line 14) and the output graph for the detected spatial hierarchy steps is returned (Line 15).

Discovering spatial hierarchy steps

In this section, we present the algorithm for *Case 2*, given in the beginning of Section 4.1, to find the *implicit* spatial hierarchy steps from QB4OLAP levels, which do not have direct (`skos:broader`) roll-up relations. In this algorithm we have to handle the situations, where there are no explicit hierarchy steps between the level members. Therefore, we benefit from schema graphs of dimensions, hierarchies, and levels for iterating through the RDF triples and compare the spatial attribute values of the level members to find the topological relations within the same dimension.

Algorithm 4 (`discoverSpatialHS`)**.** The input variables for Algorithm 4 are the schema graphs of dimensions $\mathcal{G}^S_D$, hierarchies of the dimensions $\mathcal{G}^S_{H(d)}$, levels of the hierarchies $\mathcal{G}^S_{L(h)}$, and the instance graphs of level members of levels $\mathcal{G}^I_{LM(l)}$, and attributes of level members $\mathcal{G}^I_{A(lm)}$. Each dimension $d \in D$ has a set of hierarchies $H(d)$, which is shown in the RDF graph formulation for a dimension $d \in D$ as: $\mathcal{G}^S_d = \bigcup_{h \in H(d)} \{(id^S(d)\ \texttt{qb4o:hasHierarchy}\ id^S(h))\}$. Each hierarchy $h \in H(d)$, belongs to a dimension d and has a set of levels $L(h)$, which is shown in the RDF graph formulation for a hierarchy $h \in H(d)$

Algorithm 4: discoverSpatialHS($\mathcal{G}^S_D, \mathcal{G}^S_{H(d)}, \mathcal{G}^S_{L(h)}, \mathcal{G}^I_{LM(l)}, \mathcal{G}^I_{A(lm)}$): $\mathcal{G}^I_{RU(shs)}$

Input: $\mathcal{G}^S_D$, $\mathcal{G}^S_{H(d)}$, $\mathcal{G}^S_{L(h)}$, $\mathcal{G}^I_{LM(l)}$, $\mathcal{G}^I_{A(lm)}$
Output: $\mathcal{G}^I_{RU(shs)}$

begin
  $\mathcal{G}^I_{RU(shs)} = \emptyset$; $\texttt{topoRel}_i = null$ /*initialize the output graph as an empty set and a temporary variable as null*/
  $V_{s(a_n)} = \emptyset$; $V_{s(a_k)} = \emptyset$; /*initialize temporary sets as empty sets for keeping spatial attribute values*/
  $\mathcal{G}^I_{A(lm_n)} = \emptyset$; $\mathcal{G}^I_{A(lm_k)} = \emptyset$; /*initialize empty sets to keep triple patterns for attributes of level members*/
  foreach $(id^S(d)\ \texttt{qb4o:hasHierarchy}\ id^S(h)) \in \mathcal{G}^S_D$ /*iterate through the dimensions*/ **do**
    foreach $(id^S(h)\ \texttt{qb4o:inDimension}\ id^S(d)) \in \mathcal{G}^S_H(d)$ /*iterate through the hierarchies*/ **do**
      foreach $(id^S(h)\ \texttt{qb4o:hasLevel}\ id^S(l)) \in \mathcal{G}^S_H(d)$ /*while iterating through the levels in the hiearchy, get level pairs next*/ **do**
        foreach $(id^S(l_i), id^S(l_j)) \in \mathcal{G}^S_{L(h)} \times \mathcal{G}^S_{L(h)} \mid id^S(l_i) \neq id^S(l_j) \wedge$
        $\bigcup_{lm \in LM(l)}((id^I(lm)\ \texttt{qb4o:memberOf}\ id^S(l_i)), (id^I(lm)\ \texttt{qb4o:memberOf}\ id^S(l_j))) \in \mathcal{G}^I_{LM(l)}$ /*in each level pair, while iterating through their level members, get a pair of level members $(id^I(lm_n), id^I(lm_k))$, where each level member comes from different levels*/ **do**
          foreach $(id^I(lm_n), id^I(lm_k)) \in \mathcal{G}^I_{LM(l)} \times \mathcal{G}^I_{LM(l)} \mid id^I(lm_n) \neq id^I(lm_k) \wedge id^I(lm_n) \in \mathcal{G}^I_{LM(l_i)} \implies id^I(lm_k) \in \mathcal{G}^I_{LM(l_j)} \mid \mathcal{G}^I_{LM(l_i)} \subset \mathcal{G}^I_{LM(l)} \wedge \mathcal{G}^I_{LM(l_j)} \subset \mathcal{G}^I_{LM(l)} \wedge \mathcal{G}^I_{LM(l_i)} \neq \mathcal{G}^I_{LM(l_j)}$ /*iterate through the pairs of level members*/ **do**
            foreach $((id^I(lm_n)\ id^S(a_i)\ v_{a_i}), ((id^I(lm_k)\ id^S(a_j)\ v_{a_j})) \in \mathcal{G}^I_{A(lm)} \times \mathcal{G}^I_{A(lm)}$ /*iterate through the pairs of level members' attributes*/ **do**
              $\mathcal{G}^I_{A(lm_n)} = \{(id^I(lm_n)\ id^S(a_i)\ v_{a_i})\}$; $\mathcal{G}^I_{A(lm_k)} = \{(id^I(lm_k)\ id^S(a_j)\ v_{a_j})\}$;
              $V_{s(a_n)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_n)})$;
              $V_{s(a_k)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_k)})$;
              if $V_{s(a_n)} \neq \emptyset \wedge V_{s(a_k)} \neq \emptyset$ /*make sure there are spatial values in the temporary sets*/ **then**
                foreach $(v_{a_i}, v_{a_j}) \in V_{s(a_n)} \times V_{s(a_k)}$ **do**
                  $\texttt{topoRel}_i = \texttt{relateSpatialValues}(v_{a_i}, v_{a_j})$;
                  if $\texttt{topoRel}_i \neq null$ /*make sure there is a topological relation assigned to the variable*/ **then**
                    $\mathcal{G}^I_{RU(shs)} \cup = \{(id^I(lm_n)\ \texttt{topoRel}_i\ id^I(lm_k))\}$;
  return $\mathcal{G}^I_{RU(shs)}$

as: $\mathcal{G}_h^S = \{(id^S(h)\ \texttt{qb4o:inDimension}\ id^S(d)\} \cup \bigcup_{l \in L(h)} \{(id^S(h)\ \texttt{qb4o:hasLevel}\ id^S(l))\}$. Each level l has a set of level members $LM(l) = \{lm_1, \ldots, lm_y\}$, which is shown in the RDF graph formulation for a level member $lm \in LM(l)$ as:

$\mathcal{G}_{lm}^I = \{(id^I(lm)\ \texttt{qb4o:memberOf}\ id^S(l)\}$.

Each level member lm has a set of attributes $A(lm)$. The RDF graph formulation of attributes of level members $\mathcal{G}_{A(lm)}^I$ is already given in Section 30. In Listing 2, examples of a triple pattern for level members and attributes of level members are given in Lines 1-6, Lines 8-13 and Lines 15-22, without explicit roll-up relations (Line 7).

The output of Algorithm 4 is the instance graph of roll-up relations for the *discovered* spatial hierarchy steps $\mathcal{G}_{RU(shs)}^I$ (e.g., Listing 2, Line 14). In Line 2, initially the output graph is initialized as an empty set, and a temporary variable ($\texttt{topoRel}_i$) for keeping the discovered topological relations is initialized as *null*. In Line 4, we create two temporary graphs: $\mathcal{G}_{A(lm_n)}^I$ and $\mathcal{G}_{A(lm_k)}^I$ as empty sets similar to Algorithm 3. We also create two temporary sets: $V_{s(a_n)}$ and $V_{s(a_k)}$ for storing spatial attribute values and initialize them as empty sets in Line 3.

In order to discover the spatial hierarchy steps, we need to get the attributes of all the level members from the instance graph ($\mathcal{G}_{A(lm)}^I$), and compare their spatial attribute values in pairs, where the pairs of level member attributes should be coming from two different levels in the same dimension hierarchy. Therefore, before getting the attributes of the level members, we need to classify the level members as they are grouped in different levels of a dimension hierarchy.

To achieve that, we use the schema definitions readily available in QB4OLAP, by looping through in Algorithm 4, in nested loops of dimensions in Line 5, hierarchies in the dimension (Line 6), levels in the hierarchy (Line 7). This helps us to determine the levels in a dimension hierarchy, where we can get level pairs from the same hierarchy (Line 8).

Now, while looping through the level pairs, we can identify the level members via the `qb4o:memberOf` property (Line 9). We get a pair of level members, where each level member should come from a different level, then we iterate through that pair of level members (Line 10).

Then, we get the triple patterns for the attributes of the level members from the each of the level member in the pair, and iterate through those pairs of the triple patterns (Line 11). While iterating through the triple patterns, we insert them to the temporary graphs $\mathcal{G}_{A(lm_n)}^I$ and $\mathcal{G}_{A(lm_k)}^I$ (Line 12), which are created earlier as empty sets in Line 4. So that, we can filter the spatial values from the triple patterns kept in the temporary graphs by calling the helper function `getSpatialValues` (Algorithm 1), with those input graphs $\mathcal{G}_{A(lm_n)}^I$ and $\mathcal{G}_{A(lm_k)}^I$ (Line 13).

Next, we call the helper function `getSpatialValues` (Algorithm 1) twice, with the input graphs $\mathcal{G}^{I}_{A(lm_n)}$ and $\mathcal{G}^{I}_{A(lm_k)}$. The outputs of the each (helper) function call are assigned to the temporary sets $V_{s(a_n)}$ and $V_{s(a_k)}$ correspondingly (Line 13). If these sets are not empty (Line 14), it means that `getSpatialValues` identified spatial values in the triple patterns of the input graphs.

Then, we iterate through the spatial value pairs retrieved from the each of the sets (Line 15). In this loop, we call the next helper function `relateSpatialValues` (Algorithm 2), where the input is the spatial value pairs. The output value of this function is the topological relation between the corresponding level members, and it is assigned to the initially created temporary variable $\texttt{topoRel}_i$ (Line 16).

Finally, if this $\texttt{topoRel}_i$ value is not null (Line 17), the output graph for the *spatial* hierarchy steps $\mathcal{G}^{I}_{RU(shs)}$ is incrementally generated by adding the triple pattern with the topological relation (Line 18) and the output graph for the *discovered* spatial hierarchy steps is returned in Line 19.

4.2 Factual enrichment phase

The factual enrichment phase is built around the observation facts and their spatial attributes a.k.a spatial measures and fact-dimension relations (Section 2.1).

In a QB4OLAP data structure schema, facts are linked to the dimensions at the lowest granularity level, which is the base level of the dimensions. For example, GeoFarmHerdState cube has two spatial base levels linked to the cube: Parish level and Farm level. The GeoFarmHerdState cube also has a spatial measure listed in the cube: FarmLocation (Figure F.3). In QB4OLAP, a fact schema defines the structure of a cube with the `qb:DataStructureDefinition` property (Listing 3, Line 1). Base levels (Lines 2 and 4) and measures (Line 6) are given as `qb:components` of the fact (Listing 3). The cardinality relationship between the base level and the fact can be also represented with `qb4o:cardinality` in QB4OLAP as given in Lines 2 and 4 in Listing 3.

On the other hand, with QB4SOLAP we can also represent fact-level topological relations, which are similar to the topological relations between the child-parent levels at the hierarchy steps. Fact-level topological relations are given in spatial fact schema with blue in Lines 3 and 5 (Listing 3). QB4SOLAP also extends the (cube) schema with spatial aggregate functions, which are defined over spatial measures as highlighted in blue (Listing 3, Line 7).

```
  ##Spatial Fact Schema in QB4SOLAP##
1 gfs:GeoFarmHerdState a qb:DataStructureDefinition ;
  #Lowest spatial level for each dimension in the cube#
2 qb:component [qb4o:level gfs:farm ; qb4o:cardinality qb4o:ManyToOne ;
3     qb4so:topologicalRelation qb4so:Equals] ;
4 qb:component [qb4o:level gfs:parish ; qb4o:cardinality qb4o:ManyToOne ;
```

```
5     qb4so:topologicalRelation qb4so:Within] ;
   #Example of a spatial measure in the cube#
6  qb:component [qb:measure gfs:farmLocation ;
7     qb4o:aggregateFunction qb4so:ConvexHull] .
```

Listing F.3: GeoFarmHerdState fact schema definition in QB4SOLAP

An example of an observation fact (fact member) at the instance level is given in Listing 4. A fact member is a `qb:Observation` (Line 1), which is related to the base levels (Line 2) with respect to the data structure definition (DSD) of the fact schema, and has a set of measures (Lines 3, 4) where some measures (Line 4) might have spatial values (Listing 4). In order to define a QB4OLAP fact schema, first, we need to enrich the fact members by annotating with topological relations as highlighted with blue in Line 5. We can derive topological relations between the fact members and the (base) level members by comparing the spatial measures of the fact members and spatial attributes of the (base) level members with Boolean spatial predicates. The links between the fact members and base level members are already given explicitly in Line 2 (Listing 4). However, these links are simple references between the fact and base level members, which do not describe the nature of the topological relation. By applying Boolean spatial predicates on fact and level members, we can find the exact topological relations, i.e., if a fact member *intersects* with the level member or if a fact member is *within* the level member. We explain how to detect these *explicit* fact-level (topological) relations in Section 6.

Moreover, there might be also some missing links between the (observartions) fact members and the corresponding base level members. For this case we need to find all the base level members that are spatial and derive the links between the spatial measure values and spatial attribute values (of the base level members) by using Boolean spatial predicates. We explain how to discover fact-level (topological) relations, which are not explicitly linked between observation fact and base level members in Section 20.

There are also cases that we would like to establish a direct (topological) relation between the fact members and higher granularity (parent) level members, which are not at the base level of the dimension. We explained in the example depicted in Figure F.4 that wrongly aggregating the measures (i.e., double counting) becomes a problem when we roll-up between the levels, which have many-to-many (N:M) cardinality relations (as in Parish and Drainage Area levels). Therefore, it is necessary to drill-down to the lowest granularity (fact members) and find the direct relation between the observation fact members and the corresponding level members of the higher level in many-to-many cardinality relations.

In order to prevent this problem, we address the issue in our algorithm to discover and annotate the fact-level (topological) relations, which are between the observation fact members and level members of a higher level in

an N:M cardinality relation in Section 20. For example, such a relation given is in green in Line 6 (Listing 4) that shows a topological relation between an observation fact member (farm state) and a higher level –not a base level– member (drainage area).

```
  ##GeoFarmHerdState cube:  observation fact example##
1 gfsi:farmState_103850_12_2015 a qb:Observation ;
2     gfs:farm gfsi:farm_103850 ; gfs:parish gfsi:parish_8648 ;
3     gfs:livestockUnit "4.2699999999999996"^^xsd:double ;
4     gfs:farmLocation "POINT (8.31941 56.75822)"^^geo:spatialLiteral ;
5     qb4so:equals gfsi:farm_103850 ; qb4so:within gfsi:parish_8648 ;
6     qb4so:within gfsi:water_3770 .
```

Listing F.4: GeoFarmHerdState fact member, with base levels and measures

Finally, in Section 21 we explain how to define a data structure definition (DSD) of spatial fact schema using a QB4OLAP fact schema and the spatial fact member instances derived in the previous two algorithms.

Detecting explicit fact-level relations

In this section we present an algorithm for detecting explicit fact-level topological relations between observation fact members and base level members, where there is a direct reference between the fact member and the base level member. In order to derive these topological relations we need to get the spatial attributes of fact members (spatial measures) and base level members.

Algorithm 5 (`detectFactLevelRelations`)**.** The input variables for Algorithm 5 are the instance graphs of fact members $\mathcal{G}^I_{FM(F)}$, level members $\mathcal{G}^I_{LM(l)}$, and attributes of level members $\mathcal{G}^I_{A(lm)}$.

Every fact member $f_i \in FM$ has an IRI $id^I(f_i)$ and defined as a `qb:Observation`. The RDF graph formulation of a fact member f_i is:
$\mathcal{G}^I_{f_i} = \bigcup_{l_j \in L(f_i)} \{(id^I(f_i)\ id^S(l_j)\ id^I(lm_j) \mid f_i \rightsquigarrow lm_j\} \cup$
$\bigcup_{m_k \in M(f_i)} \{(id^I(f_i)\ \ id^S(m_k)\ \ v_{m_k} \mid f_i \rightsquigarrow v_{m_k}\}$.
Here, we denote by $f_i \rightsquigarrow lm_j$ that a fact member f_i has an explicit link to a level member lm_j (e.g., Listing 4, Line 3). Remark that, we denote by $lm \rightsquigarrow v_{a_i}$ that a level member lm has value v_{a_i} for attribute a_i (Section 30), which is used in Algorithm 5, Line 12 in order to get the attribute values of the linked level members. Moreover, we denote here by $f_i \rightsquigarrow v_{m_k}$ that a fact member lm has value v_{m_k} for measure m_k (e.g., Listing 4, Lines 5 and 6). The RDF graph formulation of the other input variables are: attributes of level members $\mathcal{G}^I_{A(lm)}$ and level members $\mathcal{G}^I_{LM(l)}$ are already given, respectively, in Sections 30 and 19.

Algorithm 5: detectFactLevelRelations($\mathcal{G}^I_{FM(F)}, \mathcal{G}^I_{A(lm)}$) : $\mathcal{G}^I_{FM(F_s)}$

Input: $\mathcal{G}^I_{FM(F)}$, $\mathcal{G}^I_{A(lm)}$
Output: $\mathcal{G}^I_{FM(F_s)}$

begin
  $\mathcal{G}^I_{FM(F_s)} = \mathcal{G}^I_{FM(F)}$; $\texttt{topoRel}_i = null$; $\mathcal{G}^I_{A(f_i m_k)} = \emptyset$;
  $\mathcal{G}^I_{A(lm_j)} = \emptyset$; $V_{s(m_k)} = \emptyset$; $V_{s(a_i)} = \emptyset$; /*initialize the ouput graph, temporary variable and sets*/
  foreach /*get each observation fact (fact member)*/
  $(id^I(f_i)$ `rdf:type qb:Observation`$) \in \mathcal{G}^I_{FM(F)}$ **do**
    foreach /*get measure-level member pairs*/
    $((id^I(f_i)\ id^S(m_k)\ v_{m_k}), (id^I(f_i)\ id^S(l_j)\ id^I(lm_j)))$
    $\in \mathcal{G}^I_{FM(F)} \times \mathcal{G}^I_{FM(F)} \mid f_i \rightsquigarrow v_{m_k} \wedge lm_j \rightsquigarrow v_{a_i} \wedge$
    $(id^I(lm_j)\ id^S(a_i)\ v_{a_i}) \in \mathcal{G}^I_{A(lm)}$ /*get measure and attribute values of level members*/ **do**
      $\mathcal{G}^I_{A(f_i m_k)} = \{(id^I(f_i)\ id^S(m_k)\ v_{m_k})\}$;
      $V_{s(m_k)} =$ `getSpatialValues`$(\mathcal{G}^I_{A(f_i m_k)})$;
      if $V_{s(m_k)} \neq \emptyset$ **then**
        $\mathcal{G}^I_{A(lm_j)} = \{(id^I(lm_j)\ id^S(a_i)\ v_{a_i})\}$;
        $V_{s(a_i)} =$ `getSpatialValues`$(\mathcal{G}^I_{A(lm_j)})$;
        if $V_{s(a_i)} \neq \emptyset$ **then**
          foreach $(v_{m_k}, v_{a_i}) \in V_{s(m_k)} \times V_{s(a_i)}$ /*foreach spatial value pairs*/ **do**
            $\texttt{topoRel}_i =$ `relateSpatialValues`(v_{m_k}, v_{a_i});
            if $\texttt{topoRel}_i \neq null$ **then**
              $\mathcal{G}^I_{FM(F_s)} \cup = \{(id^I(f_i)\ \texttt{topoRel}_i\ id^I(lm_j))\}$;
  return $\mathcal{G}^I_{FM(F_s)}$

The output of Algorithm 5 is the enriched instance graph of fact members with topological relations $\mathcal{G}^I_{FM(F_s)}$. In Line 2, we initialize the output graph as the input graph of fact members (without topological relations) so that we can gradually enrich it with the detected topological relations (Line 22). Initially, the topological relation variable $\texttt{topoRel}_i$ is set to *null*. We also create two temporary graphs: $\mathcal{G}^I_{A(lm_j)}$ and $\mathcal{G}^I_{A(f_i m_k)}$ as empty sets to keep triple patterns separately in two graphs for attributes of level members and (measures of) fact members. We also create two temporary sets: $V_{s(m_k)}$ and $V_{s(a_i)}$ for keeping the spatial values from the fact and level members, and initialize them also as empty sets in Line 3.

In the first `foreach` loop (Line 4 and 5) we retrieve the observation fact members from the input graph of fact members, which corresponds to Line 1 in Listing 4. Getting the fact members allows us to access each of their measures in Line 6 and level members in Line 7 (Algorithm 5). In the next `foreach` loop (Line 9) we match the each measure-level member pair, where we can already retrieve the measure values from the input graph of fact members $\mathcal{G}^I_{FM(F)}$ (Line 10) and through the input graph for attributes of the level members $\mathcal{G}^I_{A(lm)}$ (Line 11 and 12), we can retrieve the attribute values. In Line 13, we assign the set of triples for measure attributes of fact members to a temporary graph $\mathcal{G}^I_{A(f_i m_k)}$ created earlier in Line 2. This temporary graph is given as an input to the helper function `getSpatialValues` (Algorithm 1) in Line 14 (Algorithm 5). The helper function returns the spatial attribute (measure) values of the fact members, which are kept in the temporary set $V_{s(m_k)}$. If this set is not empty (checked in Line 15) and has some spatial measures of fact member $id^I(f_i)$, we repeat the same procedure for retrieving the spatial attribute values of level member $id^I(lm_j)$ in Lines 16 and 17. If the output set for spatial attribute values $V_{s(a_i)}$ is also not empty (Line 18), then we go through the pairs of spatial values (v_{m_k}, v_{a_i}) in Line 19. In this loop, we call the next helper function `relateSpatialValues` (Algorithm 2), where the input is the spatial value pairs. The output value of this function is the topological relation between the corresponding fact and level members, which is assigned to the variable $\texttt{topoRel}_i$ (Line 20).

Discovering implicit fact-level relations

In this section, we present an algorithm for discovering fact-level (topological) relations, where there are no direct links between the fact and level members. In this algorithm we have to handle the following situations: 1) Finding the topological relations between observation facts and base level members; 2) Finding the topological relations between observation facts and parent level members in an N:M cardinality relation. In both cases there are no direct links between the observation facts and level members. Therefore, we benefit

from (QB4OLAP) schema graphs of dimensions, hierarchies, and levels for iterating through the RDF triples to distinguish the base level members, and find the parent level members, when there is an N:M cardinality relation between the levels of a hierarchy at a hierarchy step.

Algorithm 6 (`discoverFactLevelRelations`)**.** The input variables at the schema level for Algorithm 6 are the schema graphs of dimensions $\mathcal{G}^S_D$, hierarchies of the dimensions $\mathcal{G}^S_{H(d)}$, levels of the hierarchies $\mathcal{G}^S_{L(h)}$, and hierarchy steps of the hierarchies $\mathcal{G}^S_{HS(h)}$. The RDF graph formulations of the schema level input variables (dimensions $\mathcal{G}^S_{H(d)}$, hierarchies $\mathcal{G}^S_{H(d)}$, and levels $\mathcal{G}^S_{L(h)}$) are already given in Section 19. Therefore, we only explain the structure of a hierarchy step in the schema graph. Each hierarchy step hs_i is defined in the schema graph $\mathcal{G}^S_{HS(h)}$ as a blank node $\texttt{\_:hs}_i \in \mathcal{B}$. Each hierarhcy step is linked to a hierarchy $id^S(h)$ with the `qb4o:inHierarchy` predicate and has a child level $id^S(l_c)$, a parent level $id^S(l_p)$, and a cardinality relation $id^S(card)$, which are given, respectively, with `qb4o:childLevel`, `qb4o:parentLevel`, and `qb4o:pcCardinality` predicates in Line 6.

The input variables at the instance level are the instance graphs of fact members $\mathcal{G}^I_{FM(F)}$, level members of levels $\mathcal{G}^I_{LM(l)}$, and attributes of level members $\mathcal{G}^I_{A(lm)}$. We have already explained the RDF graph formulations of the instance level input variables (fact members $\mathcal{G}^I_{FM(F)}$, level members $\mathcal{G}^I_{LM(l)}$, and attributes of level members $\mathcal{G}^I_{A(lm)}$) in Section 6.

The output of Algorithm 6 is the enriched instance graph of fact members with the topological relations $\mathcal{G}^I_{FM(F_s)}$. In Line 2, we initialize the output graph as the input graph of fact members (without topological relations) so that we can gradually enrich it with the detected topological relations (Line 22). Initially, the topological relation variable $\texttt{topoRel}_i$ is set to *null*. We also create two temporary graphs: $\mathcal{G}^I_{A(lm_j)}$ and $\mathcal{G}^I_{A(f_i m_k)}$ as empty sets to keep triple patterns separately in two graphs for attributes of level members and (measures of) fact members. We also create two temporary sets: $V_{s(m_k)}$ and $V_{s(a_i)}$ for keeping the spatial values from the fact and level members, and initialize them also as empty sets in Line 3.

In order to find the topological relations between observation facts (with spatial measures) and base level members (with spatial attributes), first, we need to find all the base levels, since there is no direct link between the fact and level members. To achieve this in Algorithm 6, we use the schema definitions readily available in QB4OLAP. In Line 4, we iterate through the nested loops of dimensions to get the hierarchies and in Line 5 we iterate the nested loops of hierarchies to get the hierarchy levels. In order to find the base level of a hierarchy, we have to iterate through the hierarchy steps, where

Algorithm 6: discoverFactLevelRelations($\mathcal{G}^I_{FM(F)}, \mathcal{G}^I_{LM(l)}, \mathcal{G}^I_{A(lm)}, \mathcal{G}^S_D, \mathcal{G}^S_{H(d)}, \mathcal{G}^S_{HS(h)}$) : $\mathcal{G}^I_{FM(F_s)}$

Input: $\mathcal{G}^I_{FM(F)}, \mathcal{G}^I_{LM(l)}, \mathcal{G}^I_{A(lm)}, \mathcal{G}^S_D, \mathcal{G}^S_{H(d)}, \mathcal{G}^S_{HS(h)}$

Output: $\mathcal{G}^I_{FM(F_s)}$

begin
  $\mathcal{G}^I_{FM(F_s)} = \mathcal{G}^I_{FM(F)}$; $\texttt{topoRel}_i = null$;
  $\mathcal{G}^I_{A(f_i m_k)} = \emptyset$; $\mathcal{G}^I_{A(lm_j)} = \emptyset$; $V_{s(m_k)} = \emptyset$; $V_{s(a_i)} = \emptyset$;
  foreach $(id^S(d)$ `qb4o:hasHierarchy` $id^S(h)) \in \mathcal{G}^S_D$ **do**
    foreach $(id^S(h)$ `qb4o:inDimension` $id^S(d)) \in \mathcal{G}^S_H(d)$ **do**
      foreach $(id^S(h)$ `qb4o:hasLevel` $id^S(l_n)) \in \mathcal{G}^S_H(d)$ **do**
        foreach $(\texttt{\_:hs}_i$ `qb4o:inHierarchy` $id^S(h)) \in \mathcal{G}^S_{HS(h)} \mid$
          $(\texttt{\_:hs}_i$ `qb4o:childLevel` $id^S(l_c)) \in$
          $\mathcal{G}^S_{HS(h)} \wedge (\texttt{\_:hs}_i$ `qb4o:parentLevel` $id^S(l_p)) \in$
          $\mathcal{G}^S_{HS(h)} \wedge (\texttt{\_:hs}_i$ `qb4o:pcCardinality` $id^S(card)) \in \mathcal{G}^S_{HS(h)}$ **do**
          if $(id^S(l_n) \neq id^S(l_p)) \vee (id^S(l_n) = id^S(l_p) \wedge id^S(card) =$ `qb4o:ManyToMany`$)$ **then**
            foreach $(id^I(lm_j)$ `qb4o:memberOf` $id^S(l_n)) \in \mathcal{G}^I_{LM(l)}$ **do**
              foreach $((id^I(lm_j)$ `qb4o:memberOf` $id^S(l_n))$, $(id^I(f_i)$ `rdf:type qb:Observation`$))$
              $\in \mathcal{G}^I_{LM(l)} \times \mathcal{G}^I_{FM(F)} \mid \bigcup_{m_k \in M(f_i)} (id^I(f_i)\ id^S(m_k)\ v_{m_k}) \in$
              $\mathcal{G}^I_{FM(F)} \wedge \bigcup_{a_i \in A(lm)}$
              $(id^I(lm_j)\ id^S(a_i)\ v_{a_i}) \in \mathcal{G}^I_{A(lm)}$ **do**
                foreach
                $((id^I(f_i)\ id^S(m_k)\ v_{m_k}), (id^I(lm_j)\ id^S(a_i)\ v_{a_i})) \in$
                $\mathcal{G}^I_{FM(F)} \times \mathcal{G}^I_{A(lm)}$ **do**
                  $\mathcal{G}^I_{A(f_i m_k)} = \{(id^I(f_i)\ id^S(m_k)\ v_{m_k})\}$; $\mathcal{G}^I_{A(lm_j)} =$
                  $\{(id^I(lm_j)\ id^S(a_i)\ v_{a_i})\}$;
                  $V_{s(m_k)} = \texttt{getSpatialValues}(\mathcal{G}^I_{A(f_i m_k)})$; $V_{s(a_i)} =$
                  $\texttt{getSpatialValues}(\mathcal{G}^I_{A(lm_j)})$;
                  if $V_{s(m_k)} \neq \emptyset \wedge V_{s(a_i)} \neq \emptyset$ **then**
                    foreach $(v_{m_k}, v_{a_i}) \in V_{s(m_k)} \times V_{s(a_i)}$ **do**
                      $\texttt{topoRel}_i =$
                      $\texttt{relateSpatialValues}(v_{m_k}, v_{a_i})$;
                      if $\texttt{topoRel}_i \neq null$ **then**
                        $\mathcal{G}^I_{FM(F_s)} \cup =$
                        $\{(id^I(f_i)\ \texttt{topoRel}_i\ id^I(lm_j))\}$;
  return $\mathcal{G}^I_{FM(F_s)}$

each hierarchy step describes a child level, a parent level and a cardinality relation between the levels (Line 6). If a level $id^S(l_n)$ has never been assigned as a parent level with `qb4o:parentLevel` predicate in any of the hierarchy steps in a hierarchy h from the schema graph $\mathcal{G}^S_{HS(h)}$, then l_n is the base level of a hierarchy h (Line 7).

Thus, we can retrieve the level members of level l_n from the instance graph level members $\mathcal{G}^I_{LM(l)}$ (Line 8). In the next `foreach` loop we can pair the level members from the instance graph $\mathcal{G}^I_{LM(l)}$, and observation facts from the instance graph of fact members $\mathcal{G}^I_{FM(F)}$ (Line 9). We can retrieve a set of attributes (measures) for fact members from the fact members graph (Line 10), and a set of attributes for level members from the instance graph $\mathcal{G}^I_{A(lm)}$ (Line 11).

Then, in the next `foreach` loop in Line 12, we get the triple patterns with each measure values of the fact member and attribute values of the level member in pairs. While iterating through the (pair of) triple patterns, we insert each member of the pair to the temporary graphs for measures of fact members $\mathcal{G}^I_{A(f_i m_k)}$ and attributes of level members $\mathcal{G}^I_{A(lm_j)}$ (Line 13), which are created earlier as empty sets in Line 3. Then, we can filter the spatial values from the triple patterns kept in the temporary graphs by calling the helper function `getSpatialValues` (Algorithm 1), with those input graphs $\mathcal{G}^I_{A(f_i m_k)}$ and $\mathcal{G}^I_{A(lm_j)}$ (Line 14). We call the helper function `getSpatialValues` (Algorithm 1) twice, with the input graphs $\mathcal{G}^I_{A(f_i m_k)}$ and $\mathcal{G}^I_{A(lm_j)}$, where the outputs of the each (helper) function call are assigned to the temporary sets $V_{s(m_k)}$ and $V_{s(a_i)}$ correspondingly (Line 14). If these sets are not empty (Line 15), it means that `getSpatialValues` identified spatial values in the triple patterns of the input graphs.

Then, we iterate through the spatial value pairs retrieved from the each of the sets (Line 16). In this loop, we call the next helper function `relateSpatialValues` (Algorithm 2), where the input is a spatial value pair. The output value of this function is the topological relation between the corresponding level members, and it is assigned to the initially created temporary variable $\texttt{topoRel}_i$ (Line 17). If this $\texttt{topoRel}_i$ value is not null (Line 18), the output graph for the spatial fact members is incrementally enriched by adding the triple pattern with the topological relation (Line 19).

In order to find the topological relations between the observation facts and parent level members in an N:M cardinality relation, we check in Line 20 that if level $id^S(l_n)$ is assigned as a parent level in a hierarchy step with `qb4o:parentLevel` predicate and the hierarchy step entails an N:M relation with `qb4o:ManyToMany` predicate. If that's the case we repeat the same steps from Lines 8 to 19.

Finally, the output graph for the spatial fact members with *discovered* fact-

level (topological) relations is returned in Line 22.

Defining spatial fact DSD

In this section, we present an algorithm for re-defining the fact schema data structure definition (DSD) by enriching the DSD with fact-level topological relations. An example of a fact schema in QB4OLAP is given in the black-colored lines of Listing 3 (For now, please ignore Lines 3, 5 and 7). We re-define the spatial fact schema to QB4SOLAP (Listing 3 Lines 1-7) by using the enriched fact members that are generated via Algorithms 5 and 6.

Algorithm 7 (`defineSpatialFactDSD`)**.** The input variables for Algorithm 7 are the instance graph of spatial fact members $\mathcal{G}^I_{FM(F_s)}$ and schema graph of QB4OLAP fact schema $\mathcal{G}^S_F$. Spatial fact members in the instance graph $\mathcal{G}^I_{FM(F_s)}$ must be annotated with QB4SOLAP or can be generated by using Algorithms 5 and 6 from QB4OLAP fact members. A QB4OLAP fact schema $\mathcal{G}^S_F$ has (base) levels and measures of the cube as `qb:components` and defines the fact-level cardinality relation with `qb4o:cardinality` predicate, aggregate functions on (numerical) measures with `qb4o:aggregateFunction` predicate[11].

The output of Algorithm 7 is the enriched fact schema graph $\mathcal{G}^S_F$ annotating the fact-level relations with QB4SOLAP topological relations and measures with spatial aggregate functions.

In Line 2, we initialize the output graph as the input schema graph so that we can gradually enrich it with QB4SOLAP schema annotations (Lines 5 and 15). Initially, an aggregate function variable $\texttt{aggFunc}_i$ is created and set to *null* (Line 2).

The first `foreach` loop iterates through the fact members graph $\mathcal{G}^I_{FM(F_s)}$ and finds each fact member f_i by using the triple pattern $(id^I(f_i)$ `rdf:type qb: Observation`$)$. The second `foreach` loop gets every *distinct* topological relation $\texttt{topoRel}_i$ of the fact member f_i (Line 4). Then the output schema is annotated with the identifier of these topological relations (Line 5). Next, we get every measure v_{m_k} of the fact member f_i (Line 6), and check if it is a spatial measure (Line 7). If it is a spatial measure, we find the geometry type with *geoType* function (Line 8). We have appointed the corresponding spatial aggregate functions (Lines 10, 12, and 14) with regard to the geometry type of the spatial measure (Lines 9, 11, and 13). Finally, the output schema $\mathcal{G}^S_{F_s}$ is annotated with the identifier of these spatial aggregate functions (Line 15) and returned (Line 16).

[11] In QB4OLAP, `qb4o:AggregateFunction` class has only instances (e.g., `qb4o:Avg, qb4o:Sum` functions) for numerical measures. QB4SOLAP extends this class with a subclass `qb4so:SpatialAggregateFunction`, which has instances of spatial aggregate functions (e.g., `qb4so:ConvexHull, qb4so:Union`) for spatial measures [1, 9].

Algorithm 7: defineSpatialFactDSD($\mathcal{G}^I_{FM(F_s)}, \mathcal{G}^S_F) : \mathcal{G}^S_{F_s}$

Input: $\mathcal{G}^I_{FM(F_s)}, \mathcal{G}^S_F$
Output: $\mathcal{G}^S_{F_s}$

begin
  $\mathcal{G}^S_{F_s} = \mathcal{G}^S_F$; $\texttt{aggFunc}_i = null$; /*initalize the output graph and temporary variable*/
  foreach $(id^I(f_i)$ `rdf:type qb:Observation`$) \in \mathcal{G}^I_{FM_(F_s)}$ **do**
    foreach $(id^I(f_i)$ $\texttt{topoRel}_i$ $id^I(lm_j)) \in \mathcal{G}^I_{FM_(F_s)}$ | $\bigcup_{l_n \in L(f_i)} (id^I(f_i)$ $id^S(l_n)$ $id^I(lm_j)) \in \mathcal{G}^I_{FM_(F_s)}$ /*each $\texttt{topoRel}_i$ in the fact member triples goes into the DSD with its corresponding level l_n*/ **do**
      $\mathcal{G}^S_{F_(F_s)} \cup = \{(id^S(F)$ `qb:component [qb4o:level` $id^S(l_n)$, `qb4so:topologicalRelation` $id^S(\texttt{topoRel}_i)$`]`$)\}$;
    foreach $v_{m_k} \in (id^I(f_i)$ $id^S(m_k)$ $v_{m_k})$ /*find the spatial measures from the fact triples*/ **do**
      if v_{m_k} *is a* `geo:spatialLiteral` **then**
        switch *(geoType(v_{m_k}))* /*$geoType(v_a)$ function returns the geometry type of a given attribute value*/ **do**
          case (`POINT`) /*point geometry measures are supported to be aggregated with ConvexHull function*/ **do**
            $\texttt{aggFunc}_i =$ `qb4so:ConvexHull`
          case (`LINE`) /*line geometry measures are supported to be aggregated with Union function*/ **do**
            $\texttt{aggFunc}_i =$ `qb4so:Union`
          case (`POLYGON`) /*polygon geometry measures are supported to be aggregated with Union, Centroid,*/ **do**
            $\texttt{aggFunc}_i =$ `qb4so:Union` $\vee$ `qb4so:Centroid` $\vee$ `qb4so:MBR` /*or MBR functions*/
        $\mathcal{G}^S_{F_(F_s)} \cup = \{(id^S(F)$ `qb:component [qb:measure` $id^S(m_k)$, `qb4o:aggregateFunction` $id^S(\texttt{aggFunc}_i)$`]`$)\}$;
  return $\mathcal{G}^S_{F_s}$

5 Implementation

In this section, first we give the details on how the algorithms from Section 4 are implemented in order to generate spatially enriched RDF triples with QB4SOLAP (Sections 5.1, 5.2, 5.3, and5.4). Afterwards, we present discuss our implementation choices in Section 5.5 and present the results of applying the algorithms on the use case data – GeoFarmHerdState in Section 6 (Table F.4).

5.1 QB4SOLAP triples generation

In order to implement the algorithms given in Section 4, we have chosen a use case data set, which can be annotated with multi-dimensional concepts in QB4OLAP and has the required spatial properties to be enriched as a fully spatial multidimensional cube with QB4SOLAP. The required spatial properties are; 1) Level members in a (spatial) hierarchy must have spatial attributes, where the geometry of the attributes should be different than only a simple *point* geometry type, e.g., *polygon, line* e.t.c. Thus we can implement the hierarchical enrichment (Section 4.1). 2) The fact members should have spatial measures, thus we can implement the factual enrichment (Section 4.2).

Therefore, we have chosen GeoFarmHerdState use case, which is given as the running example throughout the paper. In Section 2, we give the spatial multi-dimensional concepts of the GeoFarmHerdState data cube and in Section 4, we give the RDF triple snippet examples of those concepts: *(a)* spatial hierarchy structure with QB4SOLAP (Listing F.1), *(b)* level members annotated with QB4OLAP and with QB4SOLAP after hierarchical enrichment (Listing F.2), *(c)* spatial fact schema (Listing F.3), and *(d)* spatial fact members with spatial measures (Listing F.4). A full overview of the GeoFarmHerdState cube with spatial and non-spatial dimensions can be found in our previous work [14] and on our project website http://extbi.cs.aau.dk/GeoFarmHerdState/.

Note that, we use the non-spatial annotation of GeoFarmHerdState data cube with QB4OLAP as an input to our algorithms, which is publicly available from our SPARQL endpoint[12] with corresponding namespaces for schema data triples[13] and instance data triples[14].

We query the endpoint and extract the RDF data in JSON format in order to use it as an input to our implementation of the four main enrichment algorithms; Algorithm 3 - `detectSpatialHS`, Algorithm 4 - `discoverSpatialHS`, Algorithm 5 - `detectFactLevel`, and Algorithm 6 - `discoverFactLevel`.

In the following, we give the implementation highlights of the each algorithm and helper function along with the code snippets.

[12]SPARQL Endpoint: http://lod.cs.aau.dk:8890/sparql
[13]QB4OLAP schema: http://extbi.cs.aau.dk/geofarm/qb4olap/farm-qb4olap-schema.ttl
[14]QB4OLAP instances: http://extbi.cs.aau.dk/geofarm/qb4olap/farm-qb4olap-input.tar.gz

5.2 Detecting explicit topological relations

Detecting explicit topological relations are addressed in the following algorithms: Algorithm 3 - `detectSpatialHS` and Algorithm 5 - `detectFactLevel`. In both cases, the source data has explicitly defined roll-up relations, which means there is a direct relation between level members with `skos:broader` for hierarchy steps (e.g., Listing F.2, Line 7) and there is a direct relation between a fact member and base level member's foreign key URI (e.g., Listing F.4, Line 2)

The input variables for Algorithm 3 - `detectSpatialHS` are the triples with roll-up relations of the hierarchy steps ($\mathcal{G}_R^I U(hs)$) and the attributes of level members ($\mathcal{G}_A^I(lm)$) from the instance data graph. Explicit `skos:broader` relations are annotated in the instance graph of hierarchy steps ($\mathcal{G}_R^I U(hs)$). Therefore, we query the endpoint, by filtering with the explicit `skos:broader` relations between all the level members. We fetch the results of the query in Node.js in JSON format.

The input variables for Algorithm 5 - `detectFactLevel` are the triples with fact members ($\mathcal{G}_F^I M(F)$) and the attributes of level members ($\mathcal{G}_A^I(lm)$) from the instance data graph. Explicit fact-level relations (by referring to the foreign key URI of level members) are annotated in the instance graph of fact members ($\mathcal{G}_F^I M(F)$). Therefore, we query the endpoint with all the fact members and the corresponding attributes of level members. We fetch the results of the query in Node.js in JSON format.

Initially, we need to provide the explicit (roll-up) relations between the level members and fact-level members to implement the Algorithms 3 and 5 for detecting the (explicit) topological relations. As mentioned in the above two paragraphs, we provide these relations from the data set by querying the endpoint and fetching the results of the query in Node.js in JSON format.

Next step is to retrieve, respectively, the spatial attribute and measure values from the attributes of the level members and fact members.

Retrieving attribute and measure values. In this step, we retrieve the (spatial) attribute values and measure values of those level members and fact members by accessing the object (o) of the each triple pattern $t = (s, p, o)$ from the instance graphs of attributes of level members ($\mathcal{G}_A^I(lm)$) and fact members ($\mathcal{G}_F^I M(F)$) as given in Listing F.5, which is followed by passing the `getLevelMemberAttributes` and `getMeasures` constants to `getSpatialValues` constant[15] as explained in the next paragraph (*filtering spatial values*) and given in Listing F.6.

[15]We differentiate measure and level attribute values in seperate constants, since a measure is annotated as `qb:MeasureProperty` and a level attribute is annotated as `qb4o:LevelAttribute` in the schema graph.

```
1 const getLevelMemberAttributes = val =>
2     val.substring (val.indexOf("(") +1,
3     val.indexOf(")"));
4 const getMeasures = mval =>
5     mval.substring (val.indexOf("(") +1,
6     mval.indexOf(")"));
```

Listing F.5: Get level member attributes and fact member measures

Filtering spatial values. Before employing spatial analysis functions, we have to filter the spatial attributes of level members and spatial measures of fact members. Spatial values are always an object (o) value in a triple pattern $t = (s, p, o)$, which is defined as spatial literals $\mathcal{L}_s$ (Section 4). Therefore, we have retrieved the attribute and measure values as objects as mentioned in the above paragraph.

We have given the helper function Algorithm 1 - `getSpatialValues`, which is used in the main algorithms. We have implemented this helper function on Node.js by filtering the WKT geometries from the input JSON data as exemplified in the following Listing F.6. We create a `locationString` constant, which takes a string value from `getLevelMemberAttributes` (Line 2). The string value is the last index location of a triple pattern constructed in `getLevelMemberAttributes`[16].

```
1 const getSpatialValues = value => {
2 const locationString =
3     getLevelMemberAttributes (value);
4 if (value.startsWith("POLYGON")) {
5     const polygons =
6        generatePolygonPoints(locationString);
7     return turf.polygon(coordinates:[polygons]); }
8 if (value.startsWith("LINE")) {
9     const lines = locationString;
10    return turf.lineString(coordinates:[lines]); }
11 if (value.startsWith("POINT")){
12    const points = locationString;
13    return turf.point(coordinates:[points]); }
14 return null; };
```

Listing F.6: Filtering spatial data types

Finding topological relations. Each of the four main enrichment algorithms (Algorithms 3, 4, 5, and 6) returns an instance graph of level members or fact members with topological relations annotated in QB4SOLAP. In order to find these topological relations, we have given a helper function in Algorithm 2 -

[16]Similarly, we create a second `locationString(2)` for spatial measure values, that takes the string value from `getMeasures`, which is not repeated in Listing F.6.

`relateSpatialValues`. This algorithm is implemented by using *boolean* functions (spatial predicates) from Turf.js library for relating spatial values and finding the appropriate topological relations. The library supports the following topological relations with corresponding predicates between certain spatial data types as given in Table F.2. A complete list of functions and details of the functions can be found online at http://turfjs.org/docs.

Table F.2: Turf.js Spatial Boolean Functions

<table>
<tr><th>EQUALS</th><th>WITHIN</th><th>INTERSECTS</th></tr>
<tr><td rowspan="3">#booleanEqual: (<code>equals</code>) between
POINT-POINT
LINE-LINE
POLYGON-POLYGON</td><td>#booleanWithin: (<code>within</code>) between
LINE-POLYGON
POLYGON-POLYGON</td><td>#booleanCrosses: (<code>crosses</code>) between
LINE-POLYGON</td></tr>
<tr><td rowspan="2">#booleanPointInPolygon: (<code>within</code>) between
POINT-POLYGON</td><td>#booleanOverlap: (<code>overlaps</code>) between
POLYGON-POLYGON</td></tr>
<tr><td>#booleanPointOnLine: (<code>intersects</code>) between
POINT-POLYGON</td></tr>
</table>

We grouped the available Turf.js spatial boolean functions in Table F.2 under three main topological relations (EQUALS, WITHIN, INTERSECTS), with respect to the simplification rules for grouping topological relations, which are given in Section 22 and explained along with Figure F.8 and Table F.1. In Table F.2, Turf.js built-in functions (predicates) are given with #boolean prefix. In parentheses, we give how we have named them in our implementation by using the corresponding built-in functions.

The following listing (Listing F.7) gives an overview of the implementation of the boolean functions from Table F.2 that are called in the main function for relating spatial values (relateSpatialValues) given in Listing F.8. We give one example for each of the main topological relations (EQUALS, WITHIN, INTERSECTS).

This *first* spatial boolean function in Listing F.7 is `equals` (Lines 1-8), which can be between any pair of the same spatial data type (Table F.2). We have grouped child level spatial (attribute) values and parent level spatial (attribute) values by their unique id (URI) for each spatial level attribute. This allows us to use *javascript array prototype (instance) methods* e.g., `every` or `some`, where we can create our own spatial predicate `equals` with condition to satisfy that `every` (grouped) child level attribute values should be equal to `every` (grouped) parent level attribute values. This ensures the multi-point, multi-line, and multi-polygon data types can be covered in our implementation.

```
  // equals function
1 const equals = (childLevelSpatialValues,
2 parentLevelSpatialValues) =>
3     childLevelSpatialValues.every(
4     childLevelSpatialValue =>
5        parentLevelSpatialValues.every(
6        parentLevelSpatialValue =>
7          turf.booleanEqual(childLevelSpatialValue,
8          parentLevelSpatialValue)));
  // within function (POLYGON-POLYGON)
9 const within = (childLevelSpatialValues,
10 parentLevelSpatialValues) => {
11     const parentLevelMultipolygonBoundingBox
12     = turf.bboxPolygon(
13      turf.bbox(
14       turf.multiPolygon(coordinates:  [
15        parentLevelSpatialValues.map(
16        parentLevelSpatialValue =>
17         pathOr([], [0], turf.getCoords(
18         parentLevelSpatialValue)))])));
  // all child level values are within the parent level
  // polygon (simplified with bounding box)
19 return childLevelSpatialValues.every(
20 childLevelSpatialValue => {
21     return turf.booleanWithin(
22      childLevelSpatialValue,
23      parentLevelMultipolygonBoundingBox);});};
  // crosses function (LINE-POLYGON)
24 const crosses = (childLevelSpatialValues,
25 parentLevelSpatialValues) =>
26     childLevelSpatialValues.some(
27     childLevelSpatialValue =>
28        parentLevelSpatialValues.some(
29        parentLevelSpatialValue =>
30          turf.booleanCrosses(childLevelSpatialValue,
31          parentLevelSpatialValue)));
```

Listing F.7: Spatial Boolean Functions

For example, in the source data, we had multi-polygons for drainage areas, where each unique drainage ares is a multi-polygon and that is composed of several polygons. Due to simplicity reasons, we did not store multi-polygon data type in RDF triples. Instead, we have annotated each unique drainage area as several polygons (of the multi-polygon), where each polygon of the drainage area is bind to its drainage area via unique id - URI of the drainage area. This means, in the instance graph of parent level members $\mathcal{G}^{I}_{A(lm_p)}$ (drainage areas), there will be triple patterns $t = (s, p, o)$, where many different polygons - objects (o) have the same subject (s) - URI of a unique drainage area to represent the multi-polygon.

In order to handle these multi-polygons, we gather them in a bound-

ing box by using `turf.bboxPolygon` and `turf.bbox` functions in Listing F.7, Lines 13-14. In Listing F.7, Lines 10-18 depicts how several polygons of the same parent level can be put into a bounding box, which is passed as a parameter to our *second* spatial boolean function `within`. Finally, the function returns in Lines 19-23 with condition to satisfy that `every` (grouped) child level attribute values should be within the simplified parent level polygon - `parentLevelMultipolygonBoundinxBox` (Line 23).

The *third* spatial boolean function in Listing F.7 is `crosses` (Lines 24-31), where we re-use the Turf.js spatial predicate `booleanCrosses`. This function is very similar to `overlaps` in implementation. The only difference is `crosses` occurs between LINE-POLYGON, `overlaps` occurs between POLYGON-POLYGON. For both cases, the condition to satisfy is that `some` of the (grouped) child level attribute values should cross/overlap `some` of the (grouped) parent level attribute values.

In the following listing (Listing F.8), we use our own spatial predicates (explained above) in order to implement the helper function Algorithm 2 - `relateSpatialValues`. Remark that we have followed the simplification rules for grouping topological relations (Figure F.8), aligned with switch cases for spatial data type pairs from Algorithm 2 in our implementation.

In our implementation given in Listing F.8, we create two functions `childLevelGeoType` (Line 3) and `parentLevelGeoType` (Line 5), which returns the geometry type of a given attribute value. This way we can implement `switch`($geoType(v_{a_c}), geoType(v_{a_p})$) `cases` from Algorithm 2 - `relateSpatialValues`.

Detecting topological relations. Finally, we have implemented detecting topological relations algorithms (Algorithms 3 and 5) with a bottom-up approach after implementing the core helper functions. In the following, we give the function implemented on Node.js for detecting topological relations (Listing F.9) between level members, which is covered in Algorithm 3. The same approach with minor differences (in parameter passing) is used in our implementation for detecting topological relations between fact-level members, which is covered in Algorithm 5.

```
1 const relateSpatialValues = (childLevelSpatialValues,
2       parentLevelSpatialValues) => {
3     const childLevelGeoType = pathOr(
4       null, [0, "geometry", "type"], childLevelSpatialValues);
5     const parentLevelGeoType = pathOr(
6       null, [0, "geometry", "type"], parentLevelSpatialValues);
7     if (childLevelGeoType === "Point" &&
8     parentLevelGeoType === "Point") {
9       if (equals(childLevelSpatialValues, parentLevelSpatialValues)) {
10       return "qb4so:equals";}
11    } else if (childLevelGeoType === "Point" &&
12      parentLevelGeoType === "LineString") {
13      if (intersects(childLevelSpatialValues, parentLevelSpatialValues)) {
14       return "qb4so:intersects";}
15    } else if (childLevelGeoType === "Point" &&
16      parentLevelGeoType === "Polygon") {
17      if (pointWithin(childLevelSpatialValues, parentLevelSpatialValues)) {
18       return "qb4so:within";}
19    } else if (childLevelGeoType === "LineString"
20      && parentLevelGeoType === "LineString") {
21      if (crosses(childLevelSpatialValues, parentLevelSpatialValues)) {
22       return "qb4so:intersects";}
23      if (overlaps(childLevelSpatialValues, parentLevelSpatialValues)) {
24       return "qb4so:overlaps";}
25    } else if (childLevelGeoType === "LineString"
26      && parentLevelGeoType === "Polygon") {
27      if (within(childLevelSpatialValues,mparentLevelSpatialValues)) {
28       return "qb4so:within";}
29      if (crosses(childLevelSpatialValues, parentLevelSpatialValues)) {
30       return "qb4so:overlaps";}
31    } else if (childLevelGeoType === "Polygon"
32      && parentLevelGeoType === "Polygon") {
33     const isWithin = within(
34      childLevelSpatialValues,
35      parentLevelSpatialValues);
36     const isOverlaps = overlaps(
37      childLevelSpatialValues, parentLevelSpatialValues);
38       if (isWithin) {
39       return "qb4so:within";}
40       if (isOverlaps) {
41       return "qb4so:overlaps";}}
42 return null;};
```

Listing F.8: Relating spatial values

Listing F.9 is constructed with the main function `detectSpatialHierarchySteps` with parameters of `parentLevelMembers` , `childLevelMembers`, and `explicit Relations`[17]. In Line 3, the constant `spatialHierachySteps` takes

[17]We do not repeat a similar listing in the paper for detecting topological relations between fact-level members (Algorithm 5) where the parameter `childLevelMembers` from Listing F.9 corresponds to fact members and `parentLevelMembers` corresponds to base level members in

the explicitRelations between child level and parent level members, and creates constants for those in Lines 6 and 7. The next step is to get the spatial values of the level members (child level members Lines 8-12 and parent level members Lines 13-17), where we utilize the helper function getSpatialValues, which is described in Listing F.6. In Line 20, we create a constant topoRel, which takes the helper function relateSpatialValues (Listing F.8) with two parameters childLevelSpatialValues and parentLevelSpatialValues that are created, in Lines 8 and 13, respectively. Next, we return the topological relations (topoRel) as predicates (p) between Lines 22-24. If a topological relation is not found we keep the explicit relation as skos:broader (Line 23). Finally, we return the new results by replacing the explicitRelations with spatialHierarchySteps (Lines 25-29).

```
1 const detectSpatialHierarchySteps = (
2   parentLevelMembers, childLevelMembers, explicitRelations) => {
3  const spatialHierarchySteps =
4   explicitRelations.results.bindings.map(
5   binding => {
6     const childLevelMemberId = binding.s.value;
7     const parentLevelMemberId = binding.o.value;
8     const childLevelSpatialValues = pathOr([ ],
9       [childLevelMemberId],childLevelMembers
10      ).map(childLevelMember =>
11       utils.getSpatialValues(
12       childLevelMember.value));
13    const parentLevelSpatialValues = pathOr([],
14      [parentLevelMemberId], parentLevelMembers
15      ).map(parentLevelMember =>
16       utils.getSpatialValues(
17       parentLevelMember.value));
18    const topoRel = utils.relateSpatialValues(
19      childLevelSpatialValues,
20      parentLevelSpatialValues);
21    return {
22       ...binding,
23       p:  {type:  "uri", value:  topoRel ||"skos:broader"}};
24 });
25    return {
26       ...explicitRelations,
27       results:  { ...explicitRelations.results,
28       bindings:  spatialHierarchySteps}
29    };
30 };
```

Listing F.9: Detecting topological relations (between level members)

We give the implementation results in Section 6.3, Table F.4 for both cases covered in Algorithms 3 and 5, together with number of input level members

the implementation of detecting topological relations between fact-level members.

and fact members.

5.3 Discovering implicit topological relations

Discovering implicit topological relations are addressed in the following algorithms: Algorithm 4 - `discoverSpatialHS` and Algorithm 6 - `discoverFactLevel`. In both cases, the source data has not any defined roll-up relations (with `skos:broader`), or has missing spatial hierarchy steps between level members. Similarly, a fact level member has no defined relation link to any spatial level member of its dimensions.

The input variables for Algorithm 4 - `discoverSpatialHS` are the triples with dimensions ($\mathcal{G}_D^S$), hierarchies in dimensions ($\mathcal{G}_H^S(d)$), levels in hierarchies ($\mathcal{G}_L^S(h)$) from the schema graph, and level members of levels ($\mathcal{G}_L^I M(l)$) and the attributes of level members ($\mathcal{G}_A^I(lm)$) from the instance data graph. Therefore, we query the endpoint by filtering with the schema elements `qb4o:hasHierarchy`, `qb4o:inDimension`, and `qb4o:hasLevel`. We fetch the results of the query in Node.js JSON format.

The input variables for Algorithm 6 - `discoverFactLe- vel` are the triples with dimensions ($\mathcal{G}_D^S$), hierarchies in dimensions ($\mathcal{G}_H^S(d)$), levels in hierarchies ($\mathcal{G}_L^S(h)$) from the schema graph, and fact members ($\mathcal{G}_F^I M(F)$), level members of levels ($\mathcal{G}_L^I M(l)$) and the attributes of level members ($\mathcal{G}_A^I(lm)$) from the instance data graph. Therefore, we query the endpoint by filtering with the schema elements `qb4o:hasHierarchy`, `qb4o:inDimension`, and `qb4o:hasLevel`. We fetch the results of the query in Node.js JSON format.

The following listing (Listing F.10) shows how we implement a schema wrapper by filtering the schema graph at our endpoint with predicates for schema elements (Lines 3, 7, 11, and 14). Once we get to the levels, we filter the level members in each level with `qb4o:memberOf` predicate (Line 11). Afterwards, we group level members by level that are in the same hierarchy and pass these grouped level members as inputs to a similar function as in Listing F.9, which is called `detectSpatialHierarchyStepsExpensive`. This function takes only two parameters without explicit relations (two sets of level members grouped by level: `parentLevelMembers` and `childLevelMembers`). We run this algorithm several times for each pairs of grouped level members (by level) within the same hierarchy as our approach is discovering implicit relations between level members and fact-level members. For fact members we use similarly one paremeter (i.e., `parentLevelMembers`) as the grouped level members (by level), and the other parameter is fact members (i.e., `childLevelMembers`), which are annotated as `qb:Observation`. In `detectSpatialHierarchyStepsExpensive` function we utilize the same helper functions that are implemented with child-parent topological relations and simplification rules defined in Section 22 along with Figure F.8 and Table F.1. This ensures to apply spatial boolean predicates (on geometries of level mem-

bers and fact members) with `relateSpatialValues` helper function only between the appropriate spatial data types given in Tables F.1 and F.2. Since, there are no explicit relations in `detectSpatialHierarchyStepsExpensive` function, `relateSpatialValues` helper function is called $Num.Of_{childLev.Mem.} \times Num.Of_{parentLev.Memb.}$ times in one iteration, where with `detectSpatialHierarchySteps` function, the helper function is called only $Num.Of_{explicitRel.}$ times.

We give the implementation results in Section 6.3, Table F.4 for both cases covered in Algorithms 4 and 6, together with number of input level members and fact members.

```
const discoverSpatialHierarchySteps = schema =>
    schema.results.bindings.filter(binding =>
      binding.p.value ==="qb4o:hasHierarchy")
    .map(hierachyBinding =>
      schema.results.bindings.filter(binding =>
      hierarchyBinding.o.value === binding.s.value
      && binding.p.value ==="qb4o:hasLevel"));
    .map(levelBinding =>
      schema.results.bindings.filter(binding =>
      levelBinding.o.value === binding.s.value
      && binding.p.value ==="qb4o:memberOf"));
const inDimension =
    schema.results.bindings.filter(binding =>
    binding.p.value === "qb4o:inDimension");
module.exports ={
    wrapper:  discoverSpatialHierarchySteps};
```

Listing F.10: Discovering topological relations (schema wrapper)

5.4 Generating fact schema

Finally, we implement the enrichment of the fact schema, based on the spatially enriched fact instances (members). In order to extract the input variables for Algorithm 7 - `defineSpatialFactDSD`, we use the spatially enriched fact members (by Algorithms 5 and 6) and non-spatial fact schema.

The first step of generating the fact schema is to look for detected and discovered topological relations between the fact and level members and then annotate each of them with `qb4so:topologicalRelation` in the fact schema as given in Listing F.3. The next step is to identify the spatial data types with helper functions `getMeasures` and `getSpatialValues` (Listings 5 and 6). Finally, for each of identified spatial data type we annotate the fact schema with the corresponding spatial aggregate function, e.g., spatial data type POINT can have *ConvexHull* aggregate function, LINE can have *Union* etc.

In our implementation of detecting and discovering topological relations between fact members and level members, we have only encountered `qb4so:`

`within` topological relation. Thus, the fact schema enrichment implementation generates Lines 4 and 5 as exemplified in Listing F.3. As spatial measures in fact members, we have found POINT spatial data type. Therefore, the fact schema enrichment implementation generates Lines 6 and 7, annotating that the spatial measure has `qb4so:ConvexHull` aggregate function, as exemplified in Listing F.3.

After the spatial enrichment is fully completed, both the schema[18] and the instance[19] data is also published from the same SPARQL endpoint with QB4SOLAP.

Table F.4 shows the results of our implementation and we discuss them in details in Qualitative Evaluation (Section 6.3).

5.5 Implementation Choices

After thoroughly describing the necessary steps and enrichment algorithms, here, we briefly present our implementation choices both in technical and strategical terms for implementing our approach

In order to answer the question: "*Can this approach be reasonably implemented on top of triple stores by directly using web and semantic web technologies?*", we have come across a number of challenges, where specific choices had to be made. These will be discussed next.

As triple store, we used: *Virtuoso version 07.20.3217 on Linux (x86_64-ubuntu-linux-gnu), Single Server Edition*. Even though, Virtuoso supports several shape types (e.g., POLYGON, MULTIPOLYGON, etc.), it has a limited number of spatial Boolean functions available as built-in functions. Therefore, we have decided to use a third party *Javascript library* for spatial analysis, which is called *Turfjs*. We implemented RDF2SOLAP on *Node.js* platform. Hardware set-up of the Node.js machine is given in Table F.3.

Table F.3: Hardware Setup (Node.js Machine)

Processor Name:	Intel Core i7
Processor Speed:	2,8 GHz
Num. of Processors:	1
Num. of Cores:	4
L2 Cache (per Core):	256 KB
L3 Cache:	6 MB
Memory:	16 GB

For each test case we have queried the triple store's SPARQL endpoint and extracted the triple data in JSON format to Node.js, which is kept in mem-

[18] http://extbi.cs.aau.dk/geofarm/qb4solap/geofarm-qb4solap-schema.ttl
[19] http://extbi.cs.aau.dk/geofarm/qb4solap/geofarm-qb4solap-output.tar.gz

ory. Using Node.js and a Javascript library for spatial analysis provides us a flexible development environment, independent from any choice of triple store.

It is mentioned earlier in Section 5.2 that we have multi-part POLYGON data (for drainage areas and parishes), which means that, when several polygons are grouped by unique (parish or water) URI they can compose a MULTIPOLYGON for a single parish or drainage area instance. From the implementation point of view, we had to implement a bounding box function for multi-part POLYGON data, in order to, call the spatial Boolean functions (within and intersects) between the correct parish and drainage area instances, then annotate the topological relations between their unique URIs. If triple stores already provided overall support of complex spatial data types, spatial indices, and a complete support of built-in spatial functions, decoupling the triple stores during development of RDF2SOLAP would not have been necessary. We could then directly use the spatial capabilities of the triple stores that were required for developing RDF2SOLAP. However, to our best knowledge, a third party spatial analysis library was needed to fully implement our RDF2SOLAP (spatial) multi-dimensional enrichment algorithms given in Section 4.

The details of our approach, endpoints, and data sets can be found on our project page[20]. The code repository for the whole implementation can be found on GitHub[21].

6 Experimental Evaluation

Table F.4: Implementation Results of Detecting and Discovering Topological Relations

		INPUT			OUTPUT		
		NumberOf Child Members	NumberOf Parent Members	NumberOf Explicit Relations	NumberOf Topological Relations		Run times (in seconds)
Section 5.2	Alg. 3	parishes: 2180	drainageAreas: 134	2683	intersects	636	29 s
					within	2046	
	Alg. 5	farmStates: 40039	parishes: 2180	39800	within	39334	7 s
Section 5.3	Alg. 4	parishes: 2180	drainageAreas: 134	NONE	intersects	1088	2622 s
					within	3392	
	Alg. 6	farmStates: 40039	parishes: 2180	NONE	within	39998	1920 s
		farmStates: 40039	drainageAreas: 134	NONE	within	39845	525 s

The rationale of developing an RDF2SOLAP enrichment module is to enrich and re-annotate existing RDF data on the Semantic Web with spatial and multi-dimensional data warehouse concepts. Then, the spatial RDF data becomes available for querying with SOLAP operations in SPARQL without losing its triple (RDF) format. We do not expect superior performance of our

[20]Project Page: http://extbi.cs.aau.dk/RDF2SOLAP
[21]RDF2SOLAP Repository: https://github.com/lopno/rdf2solap

implementation due to the limited spatial and multidimensional technologies available in the RDF/SW stack. Instead, as long as we achieve comparable performance and results, our proposal will give much more flexibility and analytical power, without needlessly spending large amounts of time on extracting and converting the data from RDF to GIS and back.

First, we present the run-times of the algorithms given in the previous section, for assessing the performance our approach, in quantitative evaluation section (Section 6.1). In order to evaluate the performance of our approach we present the total time for getting similar results over RDF data in two different (non-SW) environments in quantitative evaluation section. Next, we give the comparison baselines in Section 6.2, for describing those two different environments (GIS, RDBMS) that we are comparing our results against. Then, in qualitative evaluation section we compare our results with those two environment in terms of accuracy and coverage (Section 6.3). Finally, we share the technical lessons learned in Section 6.4 and summarize our findings in Section 6.5.

6.1 Quantitative Evaluation

The results of applying our algorithms on the running use case are summarized in Table F.4. The results show the number of topological relationships found between the level members in spatial hierarchies and between the base level members and fact members. We distinguish the results for explicit and implicit relations as implemented in the separate algorithms for spatial hierarchies (Alg. 3 and 4) and fact-level relations (Alg. 5 and 6).

The input parameters and figures for each algorithm are given in Table F.4 under the INPUT column(s). The input datasets to the algorithms are 2180 parish members, 40039 farm state members, and 134 drainage area members. The OUTPUT columns show the number of topological relations found and run times of the algorithms. In this section, we only focus on evaluating the performance of our implementation and discuss the output coverage for number of found topological relations in qualitative evaluation section.

For all the algorithms that we have implemented, we provided the test cases in the GitHub repository, where the results can be re-generated. Each experiment given in Table F.4 was run (on Node.js running) on a MacBook Pro 14,3 in a single process. The hardware details of the machine are given in Table F.3.

In Table F.4, we can see that most expensive algorithm is Alg. 4 (`discoverSpatialHS`) that runs in 2622 seconds. The algorithm takes input instances of parish and drainage area with POLYGON data type, without explicit relations as in Alg. 3 (`detecSpatialHS`). In Alg. 3, with distinct explicit relations (given 2683), the algorithm checks for the designated spatial Boolean functions (within and intersects) just 2683 times for each Boolean function. How-

Table F.5: Performance Evaluation Results (f.s.= farm states, p.= parishes, d.a.= drainage areas)

	Query Platform	Performance Results	
		Run times	Development cost
`detectSpatialHS` (p.– d.a.)	RDF2SOLAP	29 s	5 min.
	RDBMS	< 1 s	1-1.5 days
`detectFactLevelR.` (f.s. – p.)	RDF2SOLAP	7 s	5 min.
	RDBMS	< 1 s	1-1.5 days
`discoverSpatialHS` (p. – d.a.)	RDF2SOLAP	2,622	5 min.
	RDBMS	43 s	1-1.5 days
	GIS	45 s	2 days
`discoverFactLevelR.` (f.s. – p.)	RDF2SOLAP	1,920 s	5 min.
	RDBMS	95 s	1-1.5 days
	GIS	72 s	2 days
`discoverFactLevelR.` (f.s. - d.a.)	RDF2SOLAP	525 s	5 min.
	RDBMS	48 s	1-1.5 days
	GIS	41 s	2 days

ever, in Alg. 4, the algorithm calls the spatial Boolean functions (within and intersects) $134 \times 2180 = 292120$ times for each function. Similarly, compared to Alg. 5., Alg. 6 is more expensive, because of running without explicit relations, although it is much faster than Alg. 4, since the Alg. 4 calls the spatial Boolean functions between (farm states) POINT data type and POLYGON data type (for parishes and drainage areas).

In Table F.5, we compare the run times of our algorithms with two different query platforms (RDBMS and GIS) for detecting and discovering topological relations. From these platforms, Alg. 3 and 5 (to detect explicit topological relations) are only implemented on RDBMS, since GIS tool employs spatial joins instead of joining through referential integrity of explicit relations. RDBMS demonstrates great performance for both Alg. 3 and 5 by processing the queries less than 1 sec. for each. However, this query processing times do not include extracting the certain input data sets from our RDF end-point and loading the data into RDBMS and writing the SQL queries. To discover implicit topological relations in Alg. 4 and 6, GIS tool out-performed RDBMS in terms of processing time excluding the preparation and load times. RDF2SOLAP performs slower in query processing time compared to RDBMS and GIS, although, to implement the enrichment approach on native SW/RDF data, RDF2SOLAP is still the most efficient, since preparation and load time is 5 minutes and the users don't have to write SPARQL/SQL queries or know how to use a GIS tool.

Even excluding the preparation and load times, RDF2SOLAP demonstrated a comparable performance to the other query platforms, consider-

ing that the run times (in Table F.4 and F.5) for RDF2SOLAP cover also the query processing times, parsing the RDF data in JSON, calling the helper functions when necessary, and returning bounding box objects for multi-part POLYGON data.

6.2 Comparison Baselines

In order to elaborate the performance and then accuracy of our approach with choice of technologies to implement RDF2SOLAP, we compare our results (Table F.4) against two different environments: a leading GIS tool and a leading RDBMS. The software versions of the tools and hardware of the machine running these tools are given in Table F.6[22].

We load the WKT data (spatial attributes of level members) used in our experiments with the same decimal precision of the coordinates to the GIS tool, and to a geo-database on RDBMS from CSV files. We extract topological relations between the child and parent members by using spatial joins in GIS tool and using built-in spatial functions of the RDBMS.

Since both GIS tool and RDBMS cannot process RDF data in native format, we have to prepare the data to load into these environments. The preparation and load times of the data is given as development cost in Table F.5. The preparation and load time is calculated assuming that the developer has basic knowledge of the domain, extraction of RDF data with SPARQL queries, can write SQL queries, and know how to use RDBMS and GIS tools. We extracted the spatial level members (farms states, parishes, and drainage areas) used in the algorithms from our RDF endpoint, in CSV format. In order to prepare the data to be loaded into GIS tool and RDBMS we have to also use the relational schema defined by QB4SOLAP.

[22]We cannot disclose the names of the GIS tool and RDBMS tool due to license restrictions

Table F.6: Hardware and Software Setup (RDBMS Server and GIS Platform)

Hardware	
Processor Name:	Intel Core i7
Processor Speed:	2,7 - 2,9 GHz
Num. of Processors:	4
Memory:	32 GB
Software	
Operating System:	Windows 10 Enterprise (10.0) 64-bit
RDBMS Server Memory:	64-bit
RDBMS Server Memory:	16287 MB
GIS Tool Version:	64 bit 2.18.21

On the GIS tool we saved CSV data layers (for each level; farm states, parishes and drainage ares) and converted these into native GIS format which are shape files. Then we run *Join Attributes By Location* function, which is a built-in data management process. We run this function as batch process, for parishes-drainage areas (Alg. 4), farm states-parishes, and farm states-drainage areas (Alg. 6) as given in Table F.5.

Table F.7: Comparisons of number of topological relations found in each tool (f.s. = farm states, p. = parishes, d.a. = drainage areas)

		TOOLS		
		GIS	**RDBMS**	**RDF2SOLAP**
Alg. 3: (p. – d.a.)	*intersects*	N/A	1897	636
	within	N/A	785	2046
Alg. 5: (f.s.– p.)	*within*	N/A	39334	39334
Alg. 4: (p. – d.a.)	*intersects*	2556	2802	1088
	within	1039	785	3392
Alg. 6: (f.s. – p.)	*within*	39805	39984	39998
Alg. 6: (f.s. – d.a.)	*within*	39441	39845	39845

Measuring and comparing the run times (between Tables F.4 and F.5) is not the only scope of the evaluation, since algorithms in the implementation of RDF2SOLAP run in several steps with helper functions for extracting correct data (level members, hierarchy steps), finding the spatial (attribute) values etc., while on the GIS tool and the RDBMS only "*finding the topological relations*" part of the algorithms are ran. Converting RDF data into GIS and RDBMS native format, loading to these environments, and preparation of batch processes and SQL queries are inestimable manual work. Therefore, the overall cost of using offline GIS and RDBMS tools for RDF data is very expensive in terms of developer time compared to our RDF2SOLAP tool. Therefore, we use the comparison baselines for scoping out the accuracy and coverage of number of topological relations found for each algorithm in these three different environments. We give the number of topological relations in Table F.7 and discuss the results in qualitative evaluation section (Section 6.3).

6.3 Qualitative Evaluation

On the GIS tool we tested only finding (discovering) implicit topological relations (`discoverSpatialHS` and `discoverFactLevelRelations`), where we did not use direct links between farms, parishes, and drainage areas (through referential integrity of defining the explicit relations), but we used the spatial

join functionality of the (GIS data management) tool. Therefore, Alg. 3 and Alg. 5 are N/A in Table F.7 for the GIS tool.

On RDBMS, we tested both (detect and discover topological relations), where we queried with joins on the unique IDs if it was present (drainage area foreign key in parishes, parish foreign key in farm states), and with spatial joins by using `STWithin`, `STIntersects`, and `STOverlaps` built-in functions. In RDBMS, we found fewer *within* relations compared to the GIS tool and more intersects relations (Table F.7, Alg. 4). On the contrary, the RDF2SOLAP implementation finds more within relations and fewer *intersects* relations than found in the GIS tool and the RDBMS. In Alg. 3 RDF2SOLAP detects 2046 within relations, which is 47% more than RDBMS, where 785 within relations are detected. Similarly, in Alg. 4, which is also between parishes and drainage areas, RDF2SOLAP finds 47% more than GIS and 54% more than RDBMS (Table F.7). This is due to generalizing the multi-part POLYGON data as bounding boxes.

We can see that the results from RDF2SOLAP for finding implicit and explicit relations between POINT-POLYGON data types with Alg. 5 (farm states-parishes:39334) and Alg. 6 (farm states-parishes:39998 and farm states-drainage areas:39845) are very similar to the relations found in RDBMS, which can be observed in Table F.7.

We found the exact same number of topological relations (within) in RDF2SOLAP and RDBMS for Alg.5 (farm states-parishes: 39334) and Alg. 6 (farm states-drainage areas: 39845). For Alg. 6 (farm states-parishes), we found 39984 within relations in RDBMS, and 39998 within relations with only 14 difference (0,03%) in RDF2SOLAP (Table F.7). There is a little divergence between the number of (within) relations found in the GIS tool and in RDF2SOLAP in Alg. 6. There are fewer relations found in the GIS tool than RDF2SOLAP, where the difference is 193 (0,4%) for farm states-parishes and the difference is 404 (1%) for farm states-drainage areas. This can be tolerated compared to the discrepancy between the GIS tool and RDF2SOLAP (or RDBMS and RDF2SOLAP) for POLYGON-POLYGON relations.

6.4 Technical Lessons

The deviation in RDF2SOLAP for POLYGON-POLYGON relations can be prevented by using multi-part POLYGON and MULTIPOLYGON data as its original form instead of generalizing them as bounding boxes.

However, in practice, storing the multi-part POLYGON data as MULTIPOLYGONs in a triple store, loading the data to Node.js in JSON format and applying spatial Boolean functions from Turf.js library was not possible at many levels. We had encountered performance and formatting problems while loading MULTIPOLYGON data type to Virtuoso, where the debugger was not capable of providing a stack trace, where the error occurred. This

lead to missing data in the triple store for drainage areas. Even assuming that the MULTIPOLYGON data was successfully loaded, Turf.js could not handle POLYGON-MULTIPOLYGON within relations, which is normally possible on the GIS tool or on RDBMS. Since keeping the multi-part POLYGON and MULTIPOLYGON data in its original form was not feasible for the Web/Semantic Web technologies (Turf.js API and RDF Store), we had a trade-off between implementing the POLYGON-POLYGON relations in generalized bounding boxes and deviating from the results found in two other environments.

6.5 Experimental Summary

RDF2SOLAP demonstrated accurate results in comparison with two other tools (GIS and RDBMS) in terms of found topological relations between POINT-POLYGON data types. Due to generalization of multipart POLYGON relations, RDF2SOLAP has a minor divergence on the detected/discovered number of topological relations and did not meet as exact same results as in two other tools for detecting and discovering topological relations between POLYGON-POLYGON data types. In terms of productivity, RDF2SOLAP is far ahead of in-house GIS and RDBMS tools for enriching spatial RDF data by finding hierarchy steps and fact-level relations since it does not require (overall) manual work to prepare and process the data, but operates on native RDF/SW data with implicitly and explicitly defined test cases. By providing RDF2SOLAP enrichment tool, user development costs can be significantly reduced from days of preparation of data, query platforms, scripts, queries, and etc. to one time cost of pointing to the end-point and fetching the data sets from the SPARQL endpoint in test cases, which can be done in 5 minutes.

Comparing and evaluating the technical capabilities (for supporting spatial data handling) of triple stores, APIs and libraries is beyond the scope of this paper. Improvements on the underlying technologies can provide a better development environment to implement RDF2SOLAP (spatial) enrichment algorithms (Section 4) with better performance and accuracy. Further improvements to the performance (processing times) of RDF2SOLAP can be achieved either by implementing spatial indexes directly on the RDF data in triple stores or we can build an R-tree in memory on node.js using Turf.js library.

7 Related work

Utilizing DW/OLAP technologies on the Semantic Web with RDF data makes RDF data sources more easily available for interactive analysis. There has been lots of work studied OLAP and data warehousing possibilities on the

Semantic Web (SW) as summarized by Abelló et al. [17]. Our scope of work is around spatial OLAP (SOLAP) and spatial data warehouses (SDW) on the Semantic Web, which is not yet a comprehensively studied research topic. We focus on performing spatial OLAP analysis directly over multi-dimensional data published on the Semantic Web. Therefore, we review the related work with relevant approaches classified under the following titles: (1) *Data modeling and representation* (on the SW for multi-dimensional and spatial data), (2) *Metadata enrichment and MD analysis* (OLAP-like analysis over RDF data).

Data modeling and representation. The RDF Data Cube (QB) vocabulary [4] is the W3C recommendation to publish statistical data and its metadata in RDF. Thus, QB is commonly used to publish raw or already aggregated multidimensional data sets. However, QB lacks the underlying metadata for multidimensional models and OLAP operations. Set of MD concepts, such as, hierarchy levels along a cube dimension, semantics of the relationships between levels, semantics and definitions of aggregate functions are missing in QB vocabulary, which are essential in a MD schema in order to enable OLAP analysis. Therefore, Kämpgen et al. define an OLAP data model on top of QB by using SKOS [16] extensions[23] to support multi-dimensional hierarchies [18, 19]. However, the proposed model have some limitations on levels to exists only in one hierarchy. The OLAP operations are made available on the data cubes with the proposed model, but restricting the cubes with only one hierarchy per dimension. Etcheverry et al. propose QB4OLAP [7] as an extension to QB vocabulary, which supports modeling a complete MD data cube and querying the cube with OLAP operations on the Semantic Web. Modeling of MD data on the Semantic Web motivated the publication of datasets from several domains (e.g., statistical data sets from EuroStat and World Bank data, AirBase air quality data, and many other environmental and governmental open data) as RDF data cubes [20].

The need of fully multi-dimensional semantic data warehouses (where OLAP operations are enabled in SPARQL) made QB4OLAP vocabulary prominent. Therefore, RDF data cubes from statistical and environmental domains [14, 21, 22] are published with an extended QB vocabulary. Moreover, semantic Extract-Transform-Load (ETL) tools automate and ease the process of annotating and publishing open data with QB4OLAP on the Semantic Web [23]. Therefore, we can see more and more multi-dimensional datasets annotated with QB4OLAP on the Semantic Web.

These multi-dimensional semantic modeling approaches and querying with OLAP on the Semantic Web lead us to find ways for modeling, publishing and querying *spatial* data warehouses in particular, since modeling

[23] http://www.w3.org/2011/gld/wiki/ISO_Extensions_to_SKOS

and querying *spatial* data bring new challenges. QB4SOLAP [9] - a spatial extension to a fully multi-dimensional QB4OLAP vocabulary emerges the need of modeling and publishing geo-semantic data warehouses on the Semantic Web.

Modeling and publishing (non multi-dimensional) spatial data on the Semantic Web has been a focus by many communities and research groups. Some of the efforts for standardizing and aligning vocabularies to describe spatial data (e.g., locations, geometries, etc.) can be listed as GeoSPARQL [24] by the Open Geospatial Consortium (OGC), Basic Geo (WGS84 lat/long) Vocabulary by W3C Semantic Web Interest Group [25], NeoGeo Vocabularies by GeoVocab working group [26], INSPIRE Directive metadata on the Semantic Web [27], and GeoNames Ontology [28] among many others.

These standards have been commonly used in a wide scale of projects. Government Linked Data (GLD) working group listed some of these geo-vocabularies as standards to publish governmental linked data sets [29]. Andersen et al. re-use some of these vocabularies for publishing governmental and spatial data on the Semantic Web [30]. LinkedGeoData is a big contribution to the Semantic Web, which interactively transforms OpenStreetMap data to RDF data [31]. GeoKnow project focuses on linking geospatial data from heterogeneous sources [32]. More recent works by Kyzirakos et al. to transform geospatial data into RDF graphs using R2RML mappings [33] and geo-semantic labelling of open data [34] by Neumaier et al. show that spatial data on the semantic web will keep growing. However, none of these standards considers the MD aspects of spatial data for geo-semantic data warehouses.

Large volumes of spatial data on the Semantic Web yields a need for advanced modeling and analysis of such data. As mentioned earlier QB4SOLAP [9] remedy this need. Aggregate functions, cardinality relationships, topological relations are rich source of knowledge in spatial data cubes in order to query with spatial OLAP operations in SPARQL [1].

QB4ST [6] is a recent attempt to define extensions for spatio-temporal components to RDF Data Cube (QB). However, it has the inherent limitations of QB to support OLAP dimensions with hierarchies, levels and aggregate functions. Lack of OLAP hierarchies and aggregate functions in QB4ST hinders to define and operate with: topological relations at hierarchy steps, or spatial aggregate functions on spatial measures, which are essential MD concepts for SOLAP operators. These spatial MD concepts in geo-semantic data warehouses are defined together with SOLAP to SPARQL query mappings in [1].

Metadata enrichment and MD analysis. Increasing popularity of RDF data cubes and MD OLAP cubes on the Semantic Web raised interest in tools and frameworks that can ease the annotation and querying of MD data on the

Semantic Web from existing RDF sources.

Ibragimov et al. presents a framework for exploratory OLAP over Linked Open Data (LOD), where the MD schema of the data cube is annotated with QB4OLAP [35]. Based on this MD schema, they propose a system that is capable of querying data sources, extracting and aggregating data to build OLAP cubes in RDF. Similarly, Gallinucci et al. propose an exploratory OLAP approach, namely iMOLD by interactively MD modeling of linked data [36]. Their approach allows users to enrich RDF cubes with aggregation hierarchies through a user-guided process. During this interactive process, the recurring modeling patterns that express roll-up relationships between RDF concepts are recognized in the LOD, then these patterns are translated into aggregation hierarchies to enrich the RDF cube. Varga et al. enables OLAP analysis with QB2OLAP tool in [21] over statistical data published with QB vocabulary, by applying dimensional enrichment steps described thoroughly in [8]. The proposed enrichment steps allow users to enrich a QB dataset with QB4OLAP concepts such as fully-fledged dimension hierarchies. However, none of these frameworks and approaches support spatial data warehouses and SOLAP operations.

In this paper, we propose a framework, where OLAP cubes in RDF can be enriched with spatial MD concepts from *QB4SOLAP* vocabulary by employing RDF2SOLAP enrichment algorithms over QB4OLAP triples. This allows users to query MD cubes with SOLAP operators in SPARQL. Optionally, users can utilize GeoSemOLAP [2] tool on top of QB4SOLAP data sets, which helps users to formulate SOLAP queries in SPARQL.

8 Conclusion and Future Work

Motivated by the need to conciliate MD/OLAP RDF data cubes and spatial data on the Semantic Web as geo-semantic data warehouses, we have presented a number of contributions in this paper. As a first attempt to enrich RDF data cubes with spatial concepts, we have shown that the QB4SOLAP vocabulary yields the need for fully-fledged spatial data warehouse concepts (that is built on top of non-spatial QB4OLAP and RDF Data Cube (QB) vocabularies), by demonstrating the running use case examples from real world governmental open data sets from various domains (i.e., environment, farming) with complex geometry types. We give the running use case examples annotated both in QB4OLAP and QB4SOLAP vocabularies, in RDF triples and formalized the RDF triples as parameters to use in the enrichment algorithms. Second, we have built our conceptual architecture in relation to existing semantic (spatial) OLAP tools (e.g., on top of QB2OLAPem enrichment module and at the back-end of GeoSemOLAP). Third, we have provided hierarchical enrichment algorithms for two cases, which cover finding

explicit hierarchy steps with direct links between the level members and finding implicit hierarchy steps (without direct links between the level members) by comparing geometry attributes of the level members. We have defined and deployed the necessary algorithms as spatial helper functions for finding spatial attributes and comparing these attributes to derive topological relations. Fourth, we have presented the factual enrichment phase for both implicit and explicit fact-level relations between the fact and level members. Moreover, we have presented how to re-define the fact schema after the factual enrichment phase in an automated manner. Re-defining the fact schema includes also finding the spatial measures and associating them with spatial aggregate functions.

Finally, we have evaluated our experiences and the accuracy of our approach and the implementation with the underlying technologies by comparing the number of topological relations found in the RDF2SOLAP framework (between the level members in spatial hierarchies and between the level members and the fact members, respectively, during the hierarchical enrichment phase and the factual enrichment phase) against two different environments. In conclusion, RDF2SOLAP facilitates the spatial enrichment of RDF data cubes and fills an important gap in our vision of SOLAP on the Semantic Web.

Several directions are interesting for future research: creating a comprehensive benchmark test by implementing the RDF2SOLAP enrichment algorithms on different platform and testing on different use cases, deriving spatial hierarchy levels and level member instances from external geovocabularies and extending our approach in QB4SOLAP, GeoSemOLAP and RDF2SOLAP to handle highly dynamic spatio-temporal data and multi dimensional analytical queries [37]. Another line of future work would be runtime optimizations for scalable querying of spatial data warehouses [38]. Moreover, it is important to develop query optimization techniques for OLAP queries on semantic DW RDF data, similar to the ones developed for cubes and XML data [39–41]. Furthermore, to achieve scalable querying and runtime optimization, new research directions can be taken with binary serialization of the QB4SOLAP RDF data such as header dictionary triples (HDT), which is a compact data structure that can be compressed and kept in-memory, thus it enables high performance (and also concurrent) querying.

References

[1] N. Gür, T. B. Pedersen, E. Zimányi, and K. Hose, "A Foundation for Spatial Data Warehouses on the Semantic Web," *Semantic Web Journal*, vol. 9, no. 5, pp. 557–587, 2018.

[2] N. Gür, J. Nielsen, K. Hose, and T. B. Pedersen, "GeoSemOLAP: SOLAP on the Semantic Web Made Easy," in *Proceedings of the 26th International Conference Companion on World Wide Web (WWW'17)*. ACM, 2017, https://dx.doi.org/10.1145/3041021.3054731.

[3] E. Malinowski and E. Zimányi, *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Data-Centric Systems and Applications*. Springer, 2008, https://dx.doi.org/10.1007/978-3-540-74405-4.

[4] R. Cyganiak, D. Reynolds, and J. Tennison, "The RDF Data Cube Vocabulary," 2014.

[5] World Wide Web Consortium, "SPARQL Query Language for RDF," *W3C Recommendation*, 2008, https://www.w3.org/TR/sparql11-query/.

[6] Atkinson, Rob, "QB4ST: RDF Data Cube extensions for spatio-temporal components," *W3C Working Group*, 2017, https://www.w3.org/TR/qb4st/.

[7] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web using QB4OLAP," in *Data Warehousing and Knowledge Discovery (DaWaK'14)*, vol. 8646. Springer, 2014, pp. 45–56, https://dx.doi.org/10.1007/978-3-319-10160-6_5.

[8] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen, "Dimensional enrichment of statistical linked open data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 40, pp. 22–51, 2016, https://dx.doi.org/10.1016/j.websem.2016.07.003.

[9] N. Gür, K. Hose, E. Zimányi, and T. B. Pedersen, "Modeling and Querying Spatial Data Warehouses on the Semantic Web," in *Semantic Technology: 5th Joint International Semantic Technology Conference (JIST'15)*, vol. 9544. Springer, 2015, pp. 1–20, https://dx.doi.org/10.1007/978-3-319-31676-5_1.

[10] E. Malinowski and E. Zimányi, "Representing spatiality in a conceptual multidimensional model," in *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems*, ser. GIS '04. New York, NY, USA: ACM, 2004, pp. 12–22. [Online]. Available: http://doi.acm.org/10.1145/1032222.1032226

[11] M. J. Egenhofer and J. Herring, "A mathematical framework for the definition of topological relationships," in *Fourth international symposium on spatial data handling*. Zurich, Switzerland, 1990, pp. 803–813.

[12] A. Vaisman and E. Zimányi, "Spatial data warehouses," in *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[13] S. Rivest, Y. Bédard, M.-J. Proulx, M. Nadeau, F. Hubert, and J. Pastor, "SOLAP technology: Merging business intelligence with geospatial technology for interactive spatio-temporal exploration and analysis of data," *ISPRS journal of photogrammetry and remote sensing*, vol. 60, no. 1, pp. 17–33, 2005.

[14] N. Gür, K. Hose, T. B. Pedersen, and E. Zimányi, "Enabling Spatial OLAP over Environmental and Farming Data with QB4SOLAP," in *Semantic Technology: 6th Joint International Semantic Technology Conference (JIST'16)*, vol. 10055. Springer, 2016, pp. 287–304, https://dx.doi.org/10.1007/978-3-319-50112-3_22.

[15] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Computing Surveys (CSUR)*, vol. 30, no. 2, pp. 170–231, 1998.

[16] A. Miles and S. Bechhofer, "SKOS Simple Knowledge Organization System Namespace Document," *W3C Recommendation*, 2009.

[17] A. Abelló, O. Romero, T. Pedersen, R. Berlanga Llavori, V. Nebot, M. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, no. 2, pp. 571–588, 2014, https://doi.org/10.1109/TKDE.2014.2330822.

[18] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *The Semantic Web: ESWC 2012 Satellite Events*, vol. 7540. Springer, 2012, pp. 87–101, https://dx.doi.org/10.1007/978-3-662-46641-4_7.

[19] B. Kämpgen and A. Harth, "No size fits all–running the star schema benchmark with sparql and rdf aggregate views," in *Extended Semantic Web Conference*. Springer, 2013, pp. 290–304.

[20] W3C, "Data Cube Implementations," 2014, https://www.w3.org/2011/gld/wiki/Data_Cube_Implementations.

[21] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, and C. Thomsen, "QB2OLAP: Enabling OLAP on Statistical Linked Open Data," in *32nd IEEE International Conference on Data Engineering*, 2016, pp. 1346–1349.

[22] L. Galárraga, K. A. M. Mathiassen, and K. Hose, "Qboairbase: The european air quality database as an rdf cube." in *International Semantic Web Conference (Posters, Demos & Industry Tracks)*, 2017.

[23] R. P. D. Nath, K. Hose, T. B. Pedersen, and O. Romero, "Setl: A programmable semantic extract-transform-load framework for semantic data warehouses," *Information Systems*, vol. 68, pp. 17–43, 2017.

[24] M. Perry and J. Herring, "GeoSPARQL: A Geographic Query Language for RDF Data," *OGC Implementation Standard*, 2012.

[25] Brickley, Dan, "Basic Geo (WGS84 lat/long) Vocabulary," *W3C Semantic Web Interest Group*, 2003, https://www.w3.org/2003/01/geo/.

[26] Salas M., Juan and Harth, Andreas, "NeoGeo Vocabulary Specification," *GeoVocab Working Group*, 2012, http://geovocab.org/doc/neogeo/.

[27] K. Patroumpas, N. Georgomanolis, T. Stratiotis, M. Alexakis, and S. Athanasiou, "Exposing inspire on the semantic web," *Web Semantics*, vol. 35, no. P1, pp. 53–62, Dec. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2015.09.003

[28] M. Wick, "GeoNames Ontology," http://www.geonames.org/ontology/documentation.html.

[29] Hyland, Bernadette and Terrazas V., Boris, "Cookbook for open government linked data," *W3C Government Linked Data Working Group*, 2011, https://www.w3.org/2011/gld/wiki/Linked_Data_Cookbook.

[30] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Semantic Technology: 4th Joint International Semantic Technology Conference (JIST'14)*, vol. 8943. Springer, 2014, pp. 178–186, https://dx.doi.org/10.1007/978-3-319-15615-6_13.

[31] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "Linkedgeodata: A core for a web of spatial open data," *Semantic Web*, vol. 3, no. 4, pp. 333–354, 2012.

[32] G. Rojas, G. Giannopoulos, and J. J. L. Daniel Hladky, "Managing Geospatial Linked Data in the GeoKnow Project," in *The Semantic Web in Earth and Space Science. Current Status and Future Directions*, vol. 20. IOS Press, 2015, p. 51.

[33] K. Kyzirakos, D. Savva, I. Vlachopoulos, A. Vasileiou, N. Karalis, M. Koubarakis, and S. Manegold, "Geotriples: Transforming geospatial data into rdf graphs using r2rml and rml mappings," *Journal of Web Semantics*, vol. 52, pp. 16–32, 2018.

[34] S. Neumaier and A. Polleres, "Geo-semantic labelling of open data. semantics 2018-14th international conference on semantic systems," *Procedia Computer Science*, 2018.

[35] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi, "Towards exploratory olap over linked open data–a case study," in *Enabling Real-Time Business Intelligence*. Springer, 2015, pp. 114–132.

[36] E. Gallinucci, M. Golfarelli, S. Rizzi, A. Abelló, and O. Romero, "Interactive multidimensional modeling of linked data for exploratory olap," *Information Systems*, 2018.

[37] K. A. Jakobsen, A. B. Andersen, K. Hose, and T. B. Pedersen, "Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries," in *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD'15)*, 2015, http://ceur-ws.org/Vol-1426/paper-02.pdf.

[38] L. Galárraga, K. Ahlstrøm, K. Hose, and T. B. Pedersen, "Answering provenance-aware queries on rdf data cubes under memory budgets," in *The Semantic Web – ISWC 2018*, D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl, Eds. Cham: Springer International Publishing, 2018, pp. 547–565.

[39] D. Pedersen, K. Riis, and T. B. Pedersen, "Query optimization for OLAP-XML federations," in *Proceedings of the 5th International Workshop on Data Warehousing and OLAP (DOLAP'02)*. ACM, 2002, pp. 57–64.

[40] X. Yin and T. B. Pedersen, "Evaluating xml-extended olap queries based on physical algebra," *Journal of Database Management (JDM)*, vol. 17, no. 2, pp. 85–116, 2006.

[41] D. Pedersen, J. Pedersen, and T. B. Pedersen, "Integrating xml data in the targit olap system," in *Proceedings. 20th International Conference on Data Engineering*. IEEE, 2004, pp. 778–781.

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-587-1

AALBORG UNIVERSITY PRESS